What Should I Do Now?
Marrying Reinforcement Learning and Symbolic Planning

Daniel Gordon[1] Dieter Fox[1,2] Ali Farhadi[1,3]
[1]Paul G. Allen School of Computer Science, University of Washington
[2]Nvidia [3]Allen Institute for Artificial Intelligence

Abstract

Long-term planning poses a major difficulty to many reinforcement learning algorithms. This problem becomes even more pronounced in dynamic visual environments. In this work we propose Hierarchical Planning and Reinforcement Learning (HIP-RL), a method for merging the benefits and capabilities of Symbolic Planning with the learning abilities of Deep Reinforcement Learning. We apply HIP-RL to the complex visual tasks of interactive question answering and visual semantic planning and achieve state-of-the-art results on three challenging datasets all while taking fewer steps at test time and training in fewer iterations. Sample results can be found at youtu.be/0TtWJ_0mPfI[1]

1. Introduction

An important goal in developing systems with visual understanding is to create agents that interact intelligently with the world. Teaching these agents about the world requires several steps. An agent must initially learn simple behaviors such as navigation and object affordances. Then, it can combine several actions together to accomplish longer term goals. As the task complexity increases, the agent must plan farther in the future.

In recent years, researchers have predominantly trained interactive agents using either deep learning techniques on raw visual data or using planning algorithms on symbolic state representations. Deep learning has proven to be a very useful tool at learning to extract meaningful features from large sources of raw data. Deep Reinforcement Learning (Deep RL) has gained significant traction in the vision community for simple tasks like playing video games [26, 27]. Yet as task complexity increases, and longer-term planning is required, these systems can no longer learn good reactive policies due to the exponentially branching state space.

Conversely, many robotics systems still favor planning and search techniques such as RRTs and A* over the reactive Deep RL counterparts [9, 29]. The traditional planning algorithms offer better generalization when sensor data is clean, and many provide optimality guarantees which are beneficial for ensuring safety in potentially dangerous robotics environments. However most planning algorithms

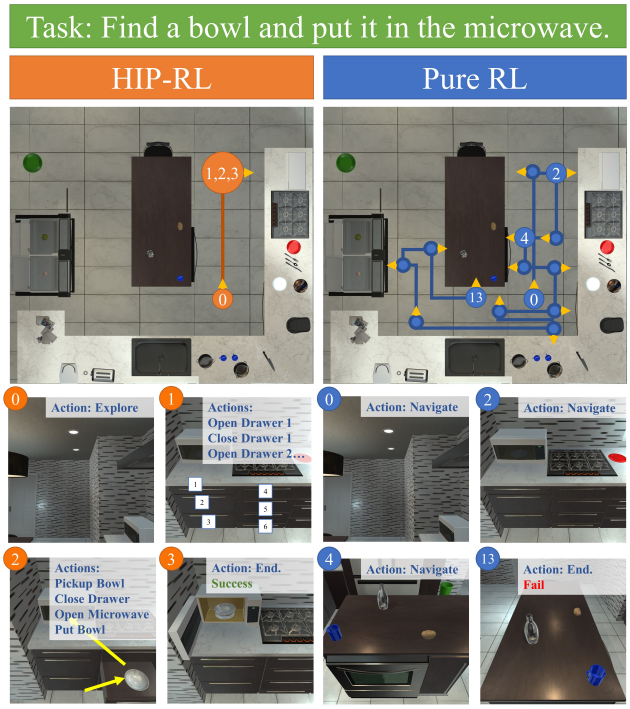

Figure 1: Sample Visual Semantic Planning task execution. The agent is asked to put a bowl in the microwave. At t=0, HIP-RL has not observed any locations where bowls can be, so it explores the room. At t=1, the Meta-Controller invokes the Planner which creates an efficient plan to check all the cabinets. It also sees the microwave and saves this information for later. At t=2 a bowl is found, and the Planner updates its plan to finish the task. Since HIP-RL already saw the microwave, it saves time by not needing an additional search. At t=3 the Planner puts the bowl in the microwave, returning control to the Meta-Controller which finishes the episode. In contrast, the Pure RL system spends much more time exploring the room, does not open any drawers, and ends the episode after many more steps, failing the task.

[1]The full dataset and code will be open sourced soon.

assume either perfect or fairly accurate state estimation and cannot operate on high dimensional raw sensor input. This is especially true for task planning algorithms which use binary values as state representations (e.g. the apple *is* in the fridge) like Planning Domain Definition Language (PDDL) solvers.

As such, deep learning and task planning have complementary benefits and drawbacks: deep learning techniques can extract information from raw (pixel) data but fail at long-term planning, and task planning require preprocessed data but can use it to construct multi-step plans which satisfy complex goals.

Much of the recent work in this area has treated learning and planning as separate problems. We present Hierarchical Planning and Reinforcement Learning (HIP-RL), a method for merging these techniques to benefit from the strengths of both. HIP-RL has several advantages over traditional planning as well as over current Deep RL techniques. 1) Due to the correctness/completeness guarantees of the planner, HIP-RL increases the accuracy and effectiveness over comparable pure RL systems. 2) By relying on a planner to create sequences of actions, HIP-RL significantly reduces the number of steps that an agent takes at test time. 3) By simplifying the learning procedure and shortening the path lengths, we are able to train our algorithm using an order of magnitude fewer training examples. 4) HIP-RL can also learn to account for noisy sensor data which may otherwise hinder a symbolic planner.

To evaluate the usefulness and effectiveness of our algorithm, we apply HIP-RL to a variety of tasks. First, we apply HIP-RL to the task of Interactive Question Answering (IQA), an extension of Visual Question Answering (VQA) where an agent dynamically navigates in and interacts with an environment in order to answer questions. We apply HIP-RL to IQUAD V1 [13] and EQA V1 [11] and achieve state-of-the-art results on both tasks. We additionally show that HIP-RL is able to perform complex Visual Semantic Planning (VSP) tasks such as `Find a bowl and put it in the microwave`, dramatically outperforming both learning-only and planning-only baselines. In general, we find that using planning and learning together results in higher accuracy, more efficient exploration, and faster training than other methods.

2. Related Work

2.1. Planning and Learning

Although both learning and planning have significant amounts of prior work, there have been relatively few attempts at merging the two. Many recent reinforcement-learning based algorithms fail when long-term planning is required; most algorithms trained on ATARI fail on the Montezuma's Revenge game due to its sparse rewards and long episodes [26]. Yet when planning and learning are combined, the results are often greater than either could do alone. One example of successfully merging planning and learning is the AlphaZero family of algorithms which combine Deep RL for board state evaluation and Monte Carlo Tree Search (MCTS) for planning and finding high-value future board positions [28]. Rather than using MCTS to intelligently explore future states, which is not feasible in a partially observed visually dynamic environment, we use the Metric-FF Planner [16] to plan a single trajectory to the goal state. This chains actions together in order to shorten the number of hierarchical decisions and reduce the size of the action space. Planning in stochastic environments is often solved using Partially Observable Markov Decision Processes. Although they are frequently used to great success [1, 6], POMDPs often assume a known noise and transition model, which is not readily applicable for algorithms which use deep feature extraction. We avoid this issue by using both planning and learning; although our planner operates under the assumption of perfect and complete information, the Meta-Controller can divert control to the learning-based methods in the event of a planner failure or to gather more information.

2.2. Hierarchical Reinforcement Learning (HRL)

HRL seeks to solve several problems with standard reinforcement learning such as handling very long episodes with sparse rewards. The design of these systems typically has one hierarchical meta-controller which invokes one of several low-level controllers. Each low-level controller is trained to accomplish a simpler task. In many cases [11, 12, 13, 22, 30] both the meta-controller and all low-level controllers are learned, and in some cases [22, 30] the tasks of the sub-controllers are also learned purely from interactions during training episodes rather than being human-engineered. This allows these systems to generalize well to unseen tasks with only a few training examples for the new tasks. In contrast, we use some learned low-level controllers, and some which use planning algorithms to directly solve subtasks. This allows our system to train qickly and still generalize well to new tasks with only moderate additional goal-state specification.

2.3. Deep RL in Virtual Visual Environments

In the past few years, many different virtual platforms have been created in order to facilitate better Deep RL. Virtual environments provide limitless labeled data and are easily parallelizable. Some of the most popular are OpenAI Gym [5] and VizDoom [19] which both build on existing video games, and MuJoCo [32] which implements more realistic contact physics. More recently, multiple environments have been created which offer near-photo-realistic and physically accurate interaction such as AI2-THOR [20], Gibson [34], and CHALET [35]. Other environments forgo photo-realism for increased rendering speed such as House3D [33], and DeepMind Lab [4].

Additionally, there have been many advancements in learning from interactions with a virtual visual environ-

ment. Recent works have used virtual environments for Question Answering [11, 12, 13], visually-driven navigation [14, 25, 37], and semantic navigation [3, 7]. However, much of the focus has been on improving navigation techniques in these environments. We are interested in expanding beyond navigation to more complex visual tasks which require both navigation and long term planning. In this work, we use the existing IQUAD V1[2] dataset presented in [13] which is built on the AI2-THOR [20] environment, and EQA V1[3] [11] which uses the House3D environment [33]. Additionally, we construct a new dataset for Visual Semantic Planning built on AI2-THOR, explained more in section 4.2.

2.4. Task and Motion Planning (TaMP)

Task and motion planning is the problem of accomplishing goals at a high level by planning the exact motion trajectories for a robot. Much of the work in TaMP uses hierarchical algorithms to plan high level actions for long-term goals and use motion planning algorithms for fine-grained motor manipulation. [18] outlines a detailed hierarchy for planning and executing simple manipulation tasks. [21] attempt to learn symbolic representations by interacting with an environment and observing the effects of actions. [8] learn heuristics to shorten the search procedure for their TaMP algorithm. We differ from these approaches in that we assume perfect manipulators, which simplifies our control setup, but use pixel-level inputs from near-photo-realistic 3D simulation environments. We also do not assume complete state information is given to the controller. Finally, we use reinforcement learning to direct our hierarchical controller, whereas most methods treat the entire state as fully observable and therefore plan and execute complete motion trajectories from the initial state.

3. Method

In order to accomplish complex tasks, a system must be able to plan long action trajectories which satisfy the task goals. To operate in a visual world, a system must learn to understand a dynamic visual environment. Thus, to learn to plan in a visual environment, we combine Deep RL with Symbolic Planning, handing control of the agent back and forth between the two methods. In this section we outline the individual components of HIP-RL as well as how they work together to solve complex learning and planning tasks.

3.1. Hierarchical Planning and Reinforcement Learning (HIP-RL)

HIP-RL consists of a hierarchical Meta-Controller, several direct (low-level) controllers, and a shared knowledge state (Figure 2). The knowledge state contains all perceptual and interactive information such as navigable locations, object positions, and which cabinets have previously been opened, as well as the goal representation. For example, in Figure 1 the knowledge state in image ❶ contains the positions of the drawers, the microwave, the red plate, and the fact that none of the drawers have been checked. Each controller can read all of the state knowledge, and can update a portion of the knowledge based on its perception; the Navigator can update the world map, and the Object Detector can update the object locations but not vice versa. At the start of an episode, the Meta-Controller chooses which direct controller should be used to advance the current state towards the goal state, invoking that direct controller with a subtask. The direct controller attempts to accomplish this subtask and returns control back to the Meta-Controller upon completion. This process is repeated until the Meta-Controller decides the full task has been accomplished and calls the Stopper to end the episode.

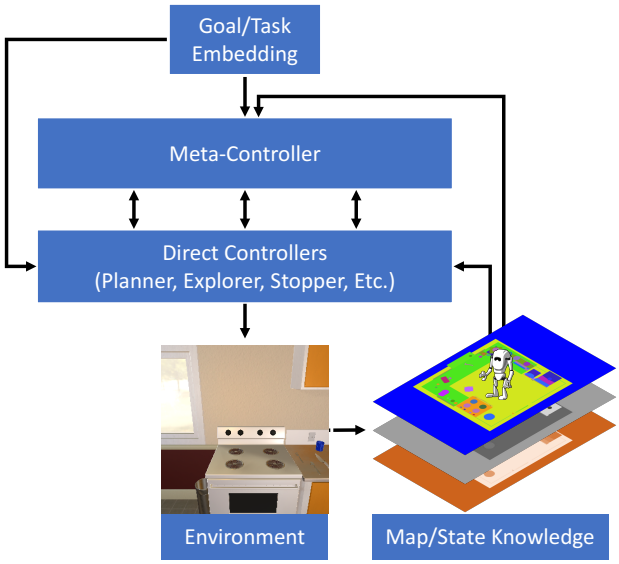

Figure 2: Diagram of the HIP-RL framework. Each direct controller interacts with the environment based off of commands given by the Meta-Controller. All controllers share a knowledge state which is updated by the various controllers during the episode based on perceptual and interactive observations.

3.2. Hierarchical Meta-Controller

The Meta-Controller receives the goal and decides which of the direct controllers to invoke. It learns to trade off between the length of an episode and the reward/penalty received for successfully or unsuccessfully ending an episode. We train this behavior using the A3C algorithm [26] to reward successful episodes, penalize unsuccessful episodes, and add a time penalty for each hierarchical step. We visualize the Meta-Controller's architecture in the context of Interactive Question Answering in Figure 3. It takes as input a top-down spatial map of object locations, the question representation, and the current image from the envi-

[2]https://github.com/danielgordon10/thor-iqa-cvpr-2018
[3]https://github.com/facebookresearch/EmbodiedQA

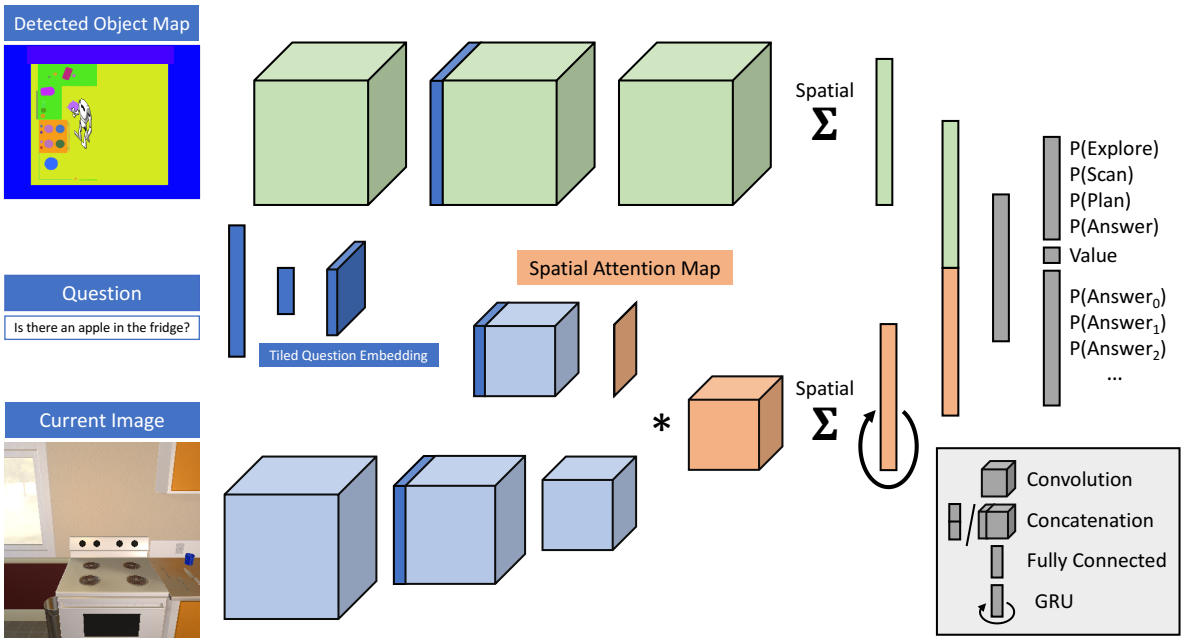

Figure 3: Overview of the network architecture for the hierarchical Meta-Controller (and answerer) used for IQA tasks. The network takes as input the full detected object map, the question, and the current image. The question is embedded and spatially tiled. We concatenate the object map features with the question embedding, perform several convolutions, and spatially sum the output. Similarly, we concatenate the image features with the question embedding, and use an attention mechanism conditioned on the question to spatially sum the features. We do not use an attention mechanism on the detected object map as this makes counting questions difficult. The image features additionally use a GRU [10] to add temporal context. Since the detected object map is purely additive (the final map contains at least as much information as the previous ones) no temporal context is necessary. The map and image features are concatenated and fed into a fully-connected layer. The network outputs a probability distribution over the actions, a value estimate for the state, and a distribution over the answers for the question.

ronment. The question embedding is concatenated with the spatial map and visual features to condition the features on the question. The network outputs a probability distribution over the action space of direct controllers as well as a value for the current state. The Meta-Controller architecture for Visual Semantic Planning is exactly the same except the question embedding is replaced with the semantic task embedding, and there is no answer branch in the output.

3.3. Planner

The Planner is tasked with returning a sequence of actions which would accomplish the goal. It operates on logical primitives using the Planning Domain Definition Language (PDDL) and is guaranteed to return a correct set of high-level planning actions given the observations are accurate. PDDL specifies states using "fluents" which can be boolean values (cabinet 1 is open, an apple is in the fridge) or numeric (location A and B are 10 steps apart). Actions consist of templatized preconditions necessary for the action to be possible, and effects, which modify the values of the fluents caused by executing that action. For example, the `Open` action takes the preconditions that an object must be openable but closed and that the agent must be near the object, and has the effect of setting the object's `closed` fluent to `False`. Goals specify a set of criteria necessary for completing some task. Even complex tasks which take hundreds of steps like `Put all the mugs into the sink.` may be easily specified. When using the Planner's output, HIP-RL sequentially takes the next Planner action, runs the Object Detector and updates the knowledge representation, and replans. This allows the Planner to easily recover from incorrect initial detections or false negatives which may be corrected over time. The Planner returns control either when the goal state has been reached, when it determines the goal is impossible, or after a fixed number of steps.

An example Planner sequence for the question `Is there a bowl in a drawer?` is as follows. The Meta-Controller invokes the Planner with the goal `All drawers have been checked or a bowl is in a drawer.` The knowledge state contains three drawers and the microwave. The Planner outputs a plan to check each drawer. The first drawer is empty, so the plan continues. In the second drawer, the bowl is found, and the Planner returns control to the Meta-Controller.

In a different example for the same task, the Planner checks each drawer and the bowl is in none of them. After all drawers are checked, the Planner returns control to

the Meta-Controller.

Grouping the multi-step output from the Planner into a single decision made by the hierarchical controller gives HIP-RL several advantages over pure RL solutions. Primarily, this significantly reduces our action space and the number of high-level steps in a single episode. Furthermore, because the Planner guarantees that the goal will be reached (or ruled impossible), HIP-RL can be more thorough and efficient in its exploration. In the examples above, if the agent had only checked a single drawer and moved on, it would have missed the opportunity to know the answer was "yes" or be more confident that the answer was "no."

In this work we use the Metric-FF PDDL solver [16], one of the most popular planning algorithms for operating on PDDL instances. Metric-FF extends the origin FF Planning algorithm [17] to both boolean and numerical values. Metric-FF uses hill-climbing on relaxed plans (plans in which contradictions are ignored) as a heuristic to estimate the distance to the goal. For numeric values, the relaxation takes the form of ignoring any non-monotonicly increasing effects that an action may have. Finding the absolute shortest solution to PDDL problems is NP-Complete, but in practice, Metric-FF usually returns nearly optimal plans in around 100 milliseconds. We include a sample PDDL state and action domain and corresponding Planner output in the supplemental material.

3.4. Object Detector

The Object Detector must detect objects from the current image, but it must also track what it has detected in the past. In this work, we assume perfect camera location knowledge which simplifies this process. The Object Detector predicts object masks as well as pixelwise depths, and the objects are projected into a global coordinate frame and merged with prior detections. In our experiments we use Mask-RCNN [15] for the detection masks and the FRCN depth estimation network [23] which are both finetuned on the training environments. We merge detections by joining the bounding cube around two detections if their 3D intersection is above a certain threshold (in practice 0.3). A more sophisticated strategy with more frequent detections such as Fusion++[24] could further improve our method (however Fusion++ requires a depth camera). In Figure 3, the Detected Object Map represents the previously detected objects and their spatial locations.

3.5. Navigator

We use the Navigator from [13] as it shows reliable performance for going to locations specified in a relative coordinate frame by a hierarchical controller. The Navigator predicts edge weights for a small region in front of the agent and uses an Egocentric Spatial GRU to update the memory state. Then it chooses the next action based on A* search to the target location. For more details, see [13]. To improve the overall execution speed of our method, our algorithm only calls the Object Detector once the navigation has finished rather than at every intermediate location.

3.6. Stopper

The Stopper is tasked with finishing an episode. For VSP, the Stopper simply terminates the episode. However for IQA, the Stopper must provide an answer to the posed question. For this, we train a network which takes the question, the entire memory state, and the current image features as input and outputs an answer. For questions from IQUAD V1, we use state information from the Object Detector to improve our accuracy. For example, for the question `Is there bread in the room?` since we track whether we have detected bread to be able to end planning upon detection, we can provide this information to the Stopper as well. For EQA V1, since the questions can be answered from a single image, we provided an additional image channel representing a detection mask of the question's subject. The Stopper is only trained based on the last state from a sequence via cross entropy over the possible answers. In practice, the Stopper shares a network with the Meta-Controller, as shown in Figure 3 which encourages the learned features to be semantically grounded.

3.7. Explorer

To gather more information about an environment, the Explorer finds a location which has not been visited and invokes the Navigator with that location. In this work, the Explorer is not learned; instead, it tracks where the agent has been and picks the location and orientation which maximizes new views while minimizing the distance from the current agent location. Note that the Explorer still operates on the Navigator's learned free-space map.

3.8. Scanner

The Scanner issues a predefined sequence of commands to the environment to obtain a 360° view of its surroundings. Specifically, it performs three full rotations at +30, 0, and -30 degrees with the horizon, stopping every 90 degrees to run the Object Detector. It is often useful to call the Scanner after calling the Explorer, but we leave this up to the hierarchical controller to learn.

4. Tasks and Environments

We focus on two tasks (Interactive Question Answering and Visual Semantic Planning) in two virtual environments (AI2-THOR [20] and House3D [33] for IQA, and AI2-THOR for VSP). Both tasks require complex visual and spatial reasoning as well as multi-step planning.

4.1. Interactive Question Answering (IQA)

We evaluate our agent on both IQUAD V1 [13] and EQA V1 [11]. IQUAD V1 provides 75,600 training questions in 25 training rooms, and 1920 test questions in 5 unseen rooms. Additionally, IQUAD V1 provides 2400 test questions in seen rooms which helps factor out errors due to

object detection noise. For EQA V1, we use the published train/val splits which consist of 7129 training questions in 648 train houses and 853 validation questions in 68 unseen houses. [11, 12] show results with the agent starting 10, 30, or 50 steps away from the goal, yet only their results for 10 steps outperform the language baselines of [31]. As such, we limit our experiments to starting 10 steps away.

In IQUAD V1, the agent must interact with kitchen scenes (opening drawers, exploring the room) in order to gather information to answer questions about the objects in the room such as `Is there a mug in the microwave?` EQA V1 places an agent in a house a certain distance away from the object of interest and asks questions like `What color is the ottoman?` Both datasets use templated language. For more detail on these datasets, see [11, 13].

For IQUAD V1, we specify the goal state for the Planner as either checking all locations where the question's subject could be, or knowing where the question's subject is. For example, for the question `Is there a mug in the microwave?` the Planner must only check the microwave, but for the question `Is there a mug in the room?` the Planner must check the microwave, the fridge, and all the drawers until it finds at least one mug. For EQA V1, we state the goal as "the agent is looking at the object of interest if it knows where it is, or it has looked through all the doorways that it has seen." This encourages the Planner to explore various different rooms in the environment in the case where the object of interest is not located in the starting room.

4.2. Visual Semantic Planning (VSP)

We also use the AI2-THOR environment [20] for semantic planning tasks such as `Put the apple in the sink.` These tasks are similar to those often done in Task and Motion Planning. Each task specifies one of 13 small objects (such as mug, fork, potato), and one of 6 target locations (such as fridge, cabinet, sink), but unlike in [36], we train one network to accomplish all tasks rather than one for each pair of object-locations. Additionally, we include navigation as a subtask of planning which [36] omits. Compared with IQA, VSP contains only a single task type but uses a larger action space to facilitate picking up and putting down objects. Although we only test a single task type, we include more complex tasks in the supplemental material. We do not perform Visual Semantic Planning in House3D [33] as the environments are static. We use the same train/test split of environments as in IQUAD V1, and construct 25,200 training tasks, 640 test tasks in unseen rooms, and 800 tasks in seen rooms. When constructing the tasks, we verify that each task is possible, yet cannot be completed by the empty plan, e.g. for the task `Put a mug in a cabinet.` there exists at least one mug and at least one cabinet, but no mugs start in cabinets. This data will be made available upon publication.

5. Experiments

We compare HIP-RL across the datasets outlined in Section 4 using existing state-of-the-art methods as baselines as well as the unimodal baselines from [31] and pure planning baselines. On each dataset, we record the accuracy/success of our method as well as the episode length. Except in the generalization experiment, all tests are done on unseen environments. [2] proposes the Success weighted by (normalized inverse) Path Length (SPL) which combines accuracy and episode length into a single metric for evaluating embodied agents on pure navigation tasks. SPL is defined as

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

where S_i is a success indicator for episode i, p_i is the path length, and ℓ_i is the shortest path length. SPL is not sufficient for question answering as an agent which never moves could still be very successful depending on the difficulty of the questions[4]. To address this issue, we propose the Shifted SPL (SSPL) metric which is defined as

$$SSPL = \frac{\mu - b}{1 - b} * SPL \tag{2}$$

where μ is the average accuracy of the method and b is the average accuracy of a baseline agent which is forced to end/answer immediately after beginning an episode. Note that SSPL directly accounts for dataset biases by subtracting the accuracy of a learned baseline rather than simply the most common answer or random chance accuracy. For the VSP experiments SPL is exactly equal to SSPL, as a baseline which cannot move will achieve 0% success. For the IQA experiments we use the "Language Only" baselines presented in [31] as b.

5.1. Baselines

On all datasets we include (at least) one pure-learning and one pure-planning baseline. The "Planner Only" baseline uses the same Plan/Act/Observe/Replan loop as HIP-RL but does not include any hierarchical decision making. Additionally, if at the start of the episode the plan is empty (for example if the agent starts looking at a wall), we rotate the agent until the plan is not empty. We also use the "Language Only" baselines from [31] which attempt to answer the question without making any actions, effectively learning the language bias of the dataset. For VSP, we introduce a "Learning Only" baseline which removes the Planner from HIP-RL and adds reward shaping to encourage certain interactions like looking at and picking up the object of the task. Even after significant training time, this method fails to learn a working policy.

[4] [31] shows significant bias exists in EQA V1 [11] and in MatterPort3D [3]

	IQUAD V1			EQA V1			VSP		
	Accuracy	Episode Length	SSPL	Accuracy	Episode Length	SSPL	Success	Episode Length	SSPL
Planner With Global Information (Shortest Path Estimate)	100%	88.710	1	100%	10	1	100%	87.477	1
State-of-the-art for IQUAD V1 [13] and EQA V1[12]	52.52%	586.890	0.015	53.58%	-	-	-	-	-
Planner Only	56.91%	**138.644**	**0.105**	49.53%	**44.424**	0.002	11.41%	**105.559**	0.059
HIP-RL	**65.99%**	357.690	0.086	**58.41%**	154.781	**0.007**	**46.01%**	427.784	**0.189**
[13] with GT Detections	64.27%	531.840	0.042	-	-	-	-	-	-
Planner Only with GT Detections	74.53%	**169.773**	**0.251**	54.15%	**22.780**	**0.025**	43.44%	**161.998**	0.245
HIP-RL With GT Detections	**81.25%**	297.238	0.177	**65.28%**	127.011	0.017	**73.75%**	254.367	**0.362**
[13] with GT Detections and Nav	69.85%	655.100	0.046	-	-	-	-	-	-
Planner Only with GT Detections and Nav	68.07%	**56.961**	0.091	49.64%	**18.784**	0.004	31.72%	**48.095**	0.268
HIP-RL with GT Detections and Nav	**83.30%**	182.191	**0.325**	**65.52%**	65.237	**0.033**	**81.88%**	161.013	**0.549**

Table 1: Comparison of accuracy and episode length with varying levels of ground truth (GT) information on unseen environments. In some cases, the "Planner Only" is fast enough to outperform HIP-RL on the SSPL metric, indicating there is still significant progress to be made on speeding up HIP-RL. Interestingly, using the navigation system instead of GT navigation helps the "Planner Only" method by giving it more varied locations to run Object Detector. The shortest path estimate for IQUAD V1 and VSP is equivalent to the "Planner Only with GT Detections and Nav" method except that it is additionally given the positions of all large objects (fridges, cabinets, etc.). In IQUAD V1 finding true shortest paths is an instance of the traveling salesman problem. For EQA V1 the shortest path is found using an oracle with preexisting knowledge of the location of the target object. In all experiments we use either the FRCN [23] depth network in conjunction with Mask-RCNN [15] or ground truth detection masks and depth.

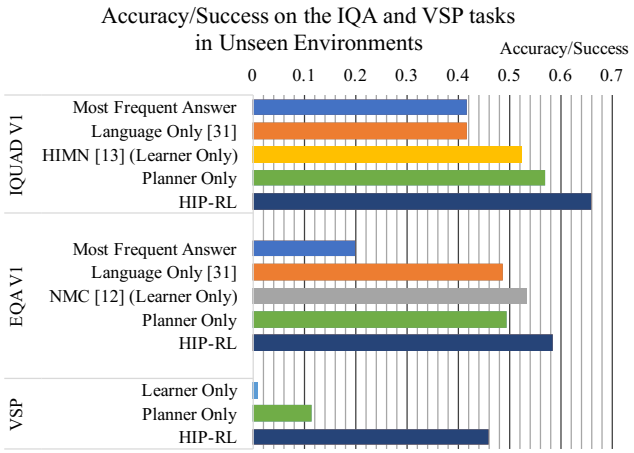

Figure 4: Accuracy of various methods on each of the tasks. In all cases, HIP-RL achieves state-of-the-art performance. We include "Learner Only" and "Planner Only" results for each experiment to show that combining both their strengths is better than either alone.

5.2. Results

We test our system for accuracy on IQUAD V1, EQA V1, and the new VSP dataset, achieving state-of-the-art performance on all tasks. The results are shown in Figure 4. In IQUAD V1 and VSP the "Planner Only" baseline outperforms the "Learning Only" baseline which coincides with the fact that the ground-truth trajectories are significantly longer and contain more necessary interactions than in EQA V1. A fundamental issue with reinforcement learning is that it must "luck into" good solutions randomly before it can improve, which can be very rare in complex multi-step tasks. Planning simplifies this by directly solving objectives rather than making guesses and observing rewards or penalties. Yet pure planning suffers from myopia in that (in our system) it assumes perfect and complete global information, leading it to ignore unobserved parts of the environment. This is most apparent in VSP where the planner assumes a task is impossible if it has not observed a location where the object can be. By combining both strategies, HIP-RL achieves the exploration tendencies of RL along with the goal-oriented direct problem solving of planning.

5.3. Ablation

We further explore the effect that various sources of inaccuracy have on HIP-RL by substituting the Object Detector and the Navigator with Ground Truth (GT) information, shown in Table 1. Adding GT detections greatly improves our accuracy across the board. This is due to all the tasks being very object-centric, so if the object is misidentified or not detected at all, the Answerer/Planner has no means of fixing the mistake. In contrast, using GT Navigation does not improve performance dramatically, but the path lengths do nearly halve. In practice we observe this is frequently not from the navigation agent wandering randomly but is instead usually from the beginning of the episodes where the map starts empty and the navigator unknowingly goes down dead ends or takes otherwise inefficient paths.

5.4. Episode Efficiency

Table 1 also lists episode lengths and SSPL scores for each method. Note that episode lengths include every interaction with the environment (turn left, open, move ahead each count as one action), not just hierarchical actions. While HIP-RL dramatically improves over [13], there is still a large gap between the shortest path estimate. Some inefficiency due to exploration is unavoidable, but there are also cases where the agent explores even after it could an-

	IQUAD V1 Unseen			IQUAD V1 Seen		
	Accuracy	Length	SSPL	Accuracy	Length	SSPL
HIP-RL	65.99%	357.690	0.086	77.75%	265.668	0.182
HIP-RL + GT Det	81.25%	297.238	0.177	87.04%	277.538	0.278
	VSP Unseen			VSP Seen		
	Success	Length	SSPL	Success	Length	SSPL
HIP-RL	46.01%	427.784	0.189	70.88%	245.301	0.384
HIP-RL + GT Det	73.75%	254.367	0.362	82.48%	170.22	0.504

Table 2: Comparison of accuracy/success on seen and unseen environments. HIP-RL is the full method, and HIP-RL + GT Det uses the ground truth detections.

swer. This generally occurs in IQUAD V1 on counting questions where the agent is not sure that it has sufficiently checked everywhere where the object could be.

5.5. Generalization

One benefit of hierarchical models is they tend to generalize better as they force certain structure to be consistent between seen and unseen environments. Yet if the hierarchical models depend on the performance of individual components, then the generalization performance of the constituent models directly affects the overall performance. In Table 2 we explore the generalization of HIP-RL on IQUAD V1 and VSP by comparing performance on rooms seen during training time with never-before-seen rooms (EQA V1 only provides test questions for unseen environments). With ground truth detections, HIP-RL generalizes quite well, losing less than 10% raw performance in both cases and staying nearly as efficient step-wise. With Mask-RCNN [15] detections and FRCN depth [23], performance is still reasonably similar, but there is a larger gap. Mask-RCNN produces high-quality results on large datasets, yet in the case of AI2-THOR [20], there are frequently only 25 training and 5 testing samples of a particular class. Thus, Mask-RCNN struggles to detect the cabinets and drawers in unseen environments from AI2-THOR (as there are many cabinets and drawers per scene but none repeat in multiple rooms), so frequently many areas remain unchecked. We believe that in scenarios with many more training examples, HIP-RL with detection would approach the same level of generalization performance as without.

5.6. Learning Speed

We compare the convergence speed of HIP-RL on the IQUAD V1 task with the previous state-of-the-art model, HIMN [13]. After only 26,000 hierarchical steps, HIP-RL with ground truth information matches the final performance of HIMN with ground truth at 8 million hierarchical steps[5]. After 120,000 hierarchical steps, HIP-RL (without ground truth) converges to its maximum performance compared to HIMN which takes 500,000 iterations and achieves significantly worse performance. HIP-RL trains orders of magnitude faster than traditional RL algorithms because the planner simplifies much of the learning by being able to immediately (upon initialization) return good, thorough trajectories. Being thorough early on is especially useful for question answering where an algorithm may get confusing feedback if it answers too soon; for example, for the question `Is there a bowl in the room?` if an agent does not see a bowl and answers "no" but the correct answer is "yes," the network will receive contradictory learning signals. By using a planner, we ensure more thorough exploration so this case is much less likely to occur, even at the beginning of training.

[5] We use the number of hierarchical steps rather than the total number of steps in the environments as hierarchical steps are of variable length and do not provide gradients to the hierarchical controller until the final low-level action.

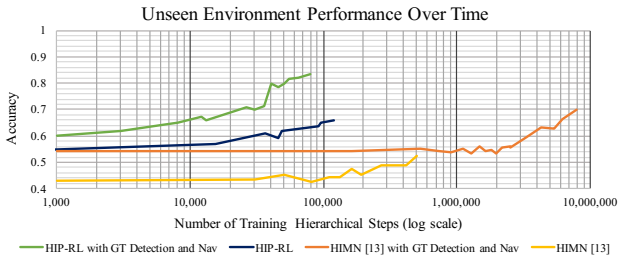

Figure 5: Learning speed of HIP-RL and HIMN [13] with and without ground truth information. HIMN's Answerer is additionally pretrained on fully observed rooms whereas HIP-RL does not require any pretraining.

6. Conclusion

In this work, we presented Hierarchical Planning and Reinforcement Learning, a method for combining the benefits of Deep Reinforcement Learning and Symbolic Planning. We demonstrate its effectiveness at increasing accuracy while simultaneously decreasing episode length and training time. Though this exact implementation may not be applicable to many other tasks, we believe the high level idea of learning to invoke various direct controllers, some of which explicitly plan, could be applied to a broader array of tasks such as Task and Motion Planning. In general, we observe that using planning algorithms to assist in performing "good" actions results in improved accuracy, test-time efficiency, and training speed. Still, HIP-RL could be improved to explore more efficiently using priors based on likely locations of an object. Additionally, we could learn the PDDL preconditions and effects directly so as to limit the need for human labels. We are excited about the potential impacts of visual agents and their ability to learn to interact more intelligently with the world around them.

7. Acknowledgements

This work was funded in part by the National Science Foundation under contract number NSF-NRI-1637479, NSF-IIS-1338054, NSF-1652052, ONR N00014-13-1-0720, the Allen Distinguished Investigator Award, and the Allen Institute for Artificial Intelligence. We would like to

thank NVIDIA for generously providing a DGX used for this research via the UW NVIDIA AI Lab (NVAIL).

References

[1] C. Amato, G. Konidaris, A. Anders, G. Cruz, J. P. How, and L. P. Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778, 2016. 2

[2] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 6

[3] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3, 6

[4] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016. 2

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. 2

[6] P. Cai, Y. Luo, D. Hsu, and W. S. Lee. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *Robotics Science and Systems (RSS) 2018*, 11, 2018. 2

[7] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *AAAI-18 Conference on Artificial Intelligence*, 2018. 3

[8] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel. Guided search for task and motion plans using learned heuristics. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 447–454. IEEE, 2016. 3

[9] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012. 1

[10] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. 4

[11] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 5, page 6, 2018. 2, 3, 5, 6

[12] A. Das, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Neural Modular Control for Embodied Question Answering. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2018. 2, 3, 6, 7

[13] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4089–4098, 2018. 2, 3, 5, 6, 7, 8

[14] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017. 3

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. 5, 7, 8

[16] J. Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003. 2, 5

[17] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 5

[18] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE, 2011. 3

[19] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016. 2

[20] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 2, 3, 5, 6, 8

[21] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. 3

[22] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016. 2

[23] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, 2016. 5, 7, 8

[24] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. Fusion++: Volumetric object-level slam. In *2018 International Conference on 3D Vision (3DV)*, pages 32–41. IEEE, 2018. 5

[25] P. W. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. *International Conference on Learning Representations (ICLR)*, 2017. 3

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016. 1, 2, 3

[27] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017. 1

[28] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton,

et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017. 2

[29] I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. 1

[30] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, page 6, 2017. 2

[31] J. Thomason, D. Gordon, and Y. Bisk. Shifting the baseline: Single modality performance on visual navigation & qa. *arXiv preprint arXiv:1811.00613*, 2018. 6

[32] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. 2

[33] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018. 2, 3, 5, 6

[34] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. 2

[35] C. Yan, D. Misra, A. Bennnett, A. Walsman, Y. Bisk, and Y. Artzi. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*, 2018. 2

[36] Y. Zhu, D. Gordon, E. Kolve, D. Fox, L. Fei-Fei, A. Gupta, R. Mottaghi, and A. Farhadi. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 483–492, 2017. 6

[37] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017. 3

Appendix A. PDDL Domain

Below is the full PDDL Domain for question answering and visual semantic planning.

```
(define (domain qa_vsp_task)
  (:requirements
    :adl
  )
  (:types
    agent
    location
    receptacle
    object
    rtype
    otype
  )

  (:predicates
    (atLocation ?a - agent ?l - location)
    (receptacleAtLocation ?r - receptacle ?l - location)
    (objectAtLocation ?o - object ?l - location)
    (openable ?r - receptacle)
    (opened ?r - receptacle)
    (inReceptacle ?o - object ?r - receptacle)
    (checked ?r - receptacle)
    (receptacleType ?r - receptacle ?t - rtype)
    (objectType ?o - object ?t - otype)
    (canContain ?t - rtype ?o - otype)
    (holds ?a - agent ?o - object)
    (holdsAny ?a - agent)
    (full ?r - receptacle)
  )

  (:functions
    (distance ?from ?to)
    (totalCost)
  )

  ;; agent goes to receptacle
  (:action GotoLocation
    :parameters (?a - agent ?lStart - location ?lEnd - location)
    :precondition (atLocation ?a ?lStart)
    :effect (and
      (atLocation ?a ?lEnd)
      (not (atLocation ?a ?lStart))
      (forall (?r - receptacle)
        (when (and (receptacleAtLocation ?r ?lEnd)
                   (or (not (openable ?r)) (opened ?r)))
          (checked ?r)
        )
      )
      (increase (totalCost) (distance ?lStart ?lEnd))
    )
  )

  ;; agent opens receptacle
  (:action OpenObject
    :parameters (?a - agent ?l - location ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (receptacleAtLocation ?r ?l)
      (openable ?r)
      (forall (?re - receptacle)
        (not (opened ?re)))
    )
    :effect (and
      (opened ?r)
      (checked ?r)
      (increase (totalCost) 1)
    )
  )

  ;; agent closes receptacle
  (:action CloseObject
    :parameters (?a - agent ?l - location ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (receptacleAtLocation ?r ?l)
      (openable ?r)
      (opened ?r)
    )
    :effect (and
      (not (opened ?r))
      (increase (totalCost) 1)
    )
  )

  ;; agent picks up object
  (:action PickupObject
    :parameters (?a - agent ?l - location ?o - object ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (objectAtLocation ?o ?l)
      (or (not (openable ?r)) (opened ?r))
      (inReceptacle ?o ?r)
      (not (holdsAny ?a))
    )
    :effect (and
      (not (inReceptacle ?o ?r))
      (holds ?a ?o)
      (holdsAny ?a)
      (increase (totalCost) 1)
    )
  )

  ;; agent puts down object
  (:action PutObject
    :parameters (?a - agent ?l - location ?ot - otype ?o - object ?r - receptacle)
    :precondition (and
      (atLocation ?a ?l)
      (receptacleAtLocation ?r ?l)
      (or (not (openable ?r)) (opened ?r))
      (not (full ?r))
      (objectType ?o ?ot)
      (holds ?a ?o)
    )
    :effect (and
      (inReceptacle ?o ?r)
      (full ?r)
      (not (holds ?a ?o))
      (not (holdsAny ?a))
      (increase (totalCost) 1)
    )
  )
)
```

Appendix B. PDDL Goal Example

Below is the goal specification for the question `Is there a mug in the room?`.

```
(:goal
  (or
    (exists (?o - object)
      (objectType ?o MugType))
    (and
      (forall (?t - rtype)
        (forall (?r - receptacle)
          (or
            (not (and (canContain ?t MugType)
                      (receptacleType ?r ?t)))
            (checked ?r)
          )
        )
      )
      (forall (?re - receptacle)
        (not (opened ?re)))
    )
  )
)
```