A Framework to Co-Optimize Robot Exploration and Task Planning in Unknown Environments

Yuanfan Xu, Zhaoliang Zhang, Jincheng Yu, Yuan Shen and Yu Wang

***Abstract*— Robots often need to accomplish complex tasks in unknown environments, which is a challenging problem, involving autonomous exploration for acquiring necessary scene knowledge and task planning. In traditional approaches, the agent first explores the environment to instantiate a complete planning domain and then invokes a symbolic planner to plan and perform high-level actions. However, task execution is inefficient since the two processes involve many repetitive states and actions. Hence, this paper proposes a framework to co-optimize robot exploration and task planning in unknown environments. To afford robot exploration and symbolic planning not being independent and separated, we design a unified structure named subtask, which is exploited to decompose the robot exploration and planning phases. To select the appropriate subtask each time, we develop a value function and a value-based scheduler to co-optimize exploration and task processing. Our framework is evaluated in a photo-realistic simulator with three complex household tasks, increasing task efficiency by 25%-29%.**

***Index Terms*—Task Planning, Reactive and Sensor-Based Planning**

I. Introduction

AUTONOMOUS robots rely on task planning to plan a sequence of high-level actions, allowing them to perform complex and multi-step tasks specified with goals, such as "having a package into a kitting box" [1] and "having a drawer closed" [2]. Symbolic planners successfully solve goal-based task planning problems, given an appropriate task specification and planning domain. The formal planning languages such as STRIPS [3] or its successor PDDL [4] are widely used to specify the domain, including a set of predicates to represent the environment states and operators with defined preconditions and effects. To generate plans and achieve its goals by symbolic planners in the real world, an agent must instantiate a PDDL planning problem with the objects and their states in the environment, typically assumed a priori fixed [5], [6], [7].

In many applications, prior environmental information is unavailable in advance, and the agent must explore the unknown environment to acquire the knowledge for planning. For example, a rescue robot must find the dangerous goods, remove them, find the trapped people, and rescue them in a cluttered and unknown environment. Additionally, a household robot has to move around and manipulate objects in a newly-arrived room, e.g., tables, boxes, and keys, to realize which and how many objects are really in the room and need rearrangement. Therefore, how to combine environmental exploration and symbolic planning is a compelling challenge for task planning in unknown environments.

Deep Reinforcement Learning (DRL) can be used to directly map from raw sensory data to actions in a partially-observable

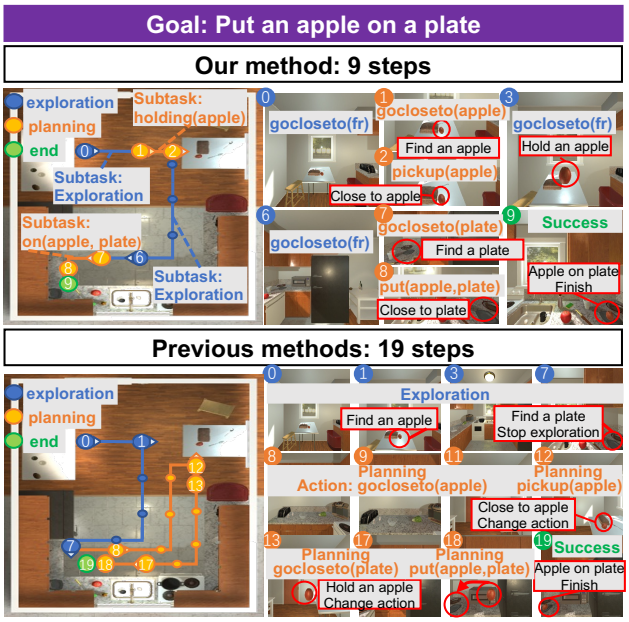


Fig. 1: A robot exploration and task planning example in unknown environments. The robot's goal is to put an apple on a plate. For our method, at t=0, the robot does not discover any apples or plates, so it chooses exploration as the current subtask and moves to the frontier. At t=1, an apple is found and the robot changes the current subtask to planning with the goal `holding(apple)` and executes its plan until t=3. At t=3, after picking up the apple, the robot explores the environment again until finding a plate. At t=7, the robot selects the subtask for planning to place the holding apple on the discovered plate and achieves the goal at t=9. On the contrary, in current methods the exploration algorithm is deployed as a single module and the symbolic planner is not invoked until exploration ends (t=7). In this case, the robot wastes much time trying to reach a state it has experienced before during exploration, such as `closeto(apple)` at t=1 and t=12, `closeto(plate)` at t=7 and t=18, which results in low efficiency.

Manuscript received: June 10, 2022; Revised: September 1, 2022; Accepted: October 1, 2022. This paper was recommended for publication by Editor J. Yi upon evaluation of the Associate Editor and Reviewers' comments.

The authors are with the Department of Electronic Engineering, Tsinghua University, Beijing, China. xuyf20@mails.tsinghua.edu.cn, {yu-jc, yu-wang}@tsinghua.edu.cn.

This research was supported by National Natural Science Foundation of China (U20A20334, U19B2019 and M-0248), Tsinghua-Meituan Joint Institute for Digital Life, Tsinghua EE Independent Research Project, Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips.

Digital Object Identifier (DOI): see top of this page.

world and an end-to-end manner [8], [9]. Nevertheless, these methods cannot exploit the state-of-the-art autonomous exploration approaches [10] and the power of symbolic planners [11]. Hierarchical Reinforcement Learning (HRL) solutions [12], [13] regard the robot explorer and planner as different low-level policies and adopt a high-level meta-controller to decide which of the policies to invoke. The main drawback of RL-based methods is that they need to be designed and trained for each specific task type, and can not generalize over different tasks.

An alternative and domain-independent solution couples the exploration and planning processes in chronological order [14]. In this scheme, the agent explores the unknown environment until discovering all the objects needed to satisfy the goal and then invokes a symbolic planner to solve the problem. However, the agent repeatedly experiences some of the same states in the two independent processes. As illustrated in Figure 1, the task goal is to put an apple on a plate. The robot moves close to an apple to find and locate it during exploration. Meanwhile, the symbolic planner also needs the robot to be close to the apple and pick it up. Therefore, the robot moves along the repeated trajectory and returns to an already experienced state, which results in inefficiency.

This paper proposes a framework that allows an agent to explore and plan simultaneously rather than in chronological order to co-optimize exploration and symbolic planning and eliminate the repeated states. We decompose a complex robot task planning problem that needs all knowledge to be solved into a series of simple subtasks, so that the robot can accomplish the subtasks with part of information during exploration. The robot exploration problem is also decomposed into subtasks, which consist of selecting a frontier (areas that separate the free from the unknown regions) and navigating to the corresponding location. The subtask integrates robot exploration and task planning at the symbolic level so that the two independent processes can be co-optimized with a unified value function. Every time the robot arrives at a new state, a designed Subtask Manager (Section IV-D) updates the currently available subtasks and their values. Then a Value-based Scheduler (Section IV-E) selects the appropriate subtask contributing the most to the goal.

The contributions of this paper are summarized as follows:

- We propose a framework to co-optimize robot exploration and symbolic planning, affording the agent to efficiently solve complex tasks in unknown environments.
- We propose an approach that extracts subtasks from robot exploration and symbolic planning to unify the two originally independent processes. Additionally, we develop a value-based scheduling strategy to co-optimize information acquisition and task processing.
- We evaluate our framework on three tasks in a photo-realistic simulator and improve execution efficiency by about 29%.

II. Related Work

In the field of robotics, task planning or symbolic planning methods are widely used to solve complex tasks [1], [15]. ROSPlan [16] and Plansys2 [17] are two popular planning systems for robotics. They load the PDDL domain and problem from files to generate the plan and encapsulate plan execution. However, these general planning frameworks are only suitable for solving tasks where the planning domain and problem are available from the beginning.

There is a great deal of work related to integrated task and motion planning (TaMP), planning under uncertainty, and hierarchical planning. Most of them [18], [19], [20] assume that the task-level planning domain is fixed and known to the robot, and the uncertainty results from the motion-level changes and constraints. However, we focus on solving task planning problems where the grounded task-level planning domain is initially unknown and try to find a balance between robot exploration and task planning.

Various RL-based methods can address the problem of goal-based symbolic planning in unknown environments. For example, Garnelo *et al.* [8] designed a neural back-end to update the abstract domain online every time the robot discovers new objects and a symbolic front-end to output the current action. Both HRL4IN [13] and HIP-RL [12] employ a hierarchical RL architecture. HRL4IN exploited a high-level policy to create subgoals for the low-level policy and capabilities (navigation, manipulation, or both) that the low-level policy is allowed to use. And HIP-RL consists of several predefined low-level controllers, e.g., planner, explorer, or detector, and only trains the high-level controller to decide which low-level controller is invoked. However, these RL-based methods can only solve a single task type as their inputs and rewards are domain-specific. Besides, collecting training data tends to be expensive in robot manipulation problems, which limits these methods to solve simple tasks. On the contrary, our approach is domain-independent and can solve different complex tasks.

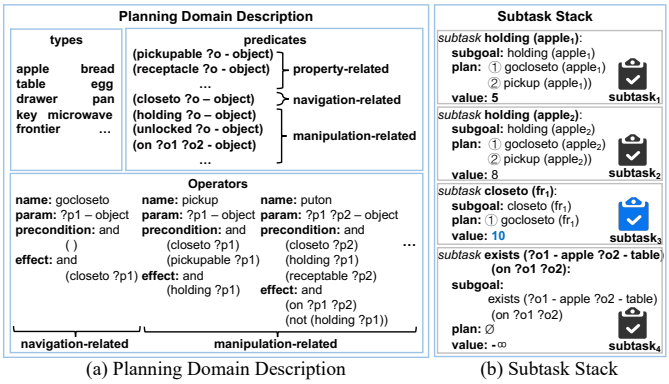
OGAMUS [14] is a domain-independent framework for robot task planning in unknown environments. It treats robot exploration and planning as independent parts, where the agent first explores and then plans. The sequential combination suffers from repeated states and paths and is inefficient. Unlike OGAMUS, our framework integrates robot exploration with task planning at the symbolic level and co-optimizes the two processes simultaneously.

III. Preliminary

We start by introducing the necessary notations and background of symbolic planning, and the problem formulation on applying symbolic planning in robotics scenarios.

Let $\mathcal{P}$ be a set of first order predicates, $\mathcal{T}$ a set of object types, $\mathcal{V}$ a set of parameters (also called variables), and $\mathcal{C}$ a set of object entities (also called constants). Each element of $\mathcal{V}$ and $\mathcal{C}$ belongs to a specific type $t \in \mathcal{T}$. $\mathcal{P}(\mathcal{V})$ denotes the set of atoms $P(v_1, ..., v_n)$, where $P \in \mathcal{P}$ and $v_i \in \mathcal{V}$, and $\mathcal{P}(\mathcal{C})$ denotes the set of facts, obtained by grounding $\mathcal{P}(\mathcal{V})$ with the real object entities in $\mathcal{C}$. The state of the world $s \in \mathcal{P}(\mathcal{C})$ is described by the set of facts that are true in it.

We use $\mathcal{O}$ to denote a set of operators, corresponding to the basic actions that the robot can perform. An operator $op \in \mathcal{O}$ is a tuple $\langle \texttt{par}(op), \texttt{pre}(op), \texttt{eff}^+(op), \texttt{eff}^-(op) \rangle$, where



(a) Planning Domain Description (b) Subtask Stack

Fig. 2: (a) Example of domain description $\mathcal{D}$. Predicates and operators are categorized into property-related, navigation-related, and manipulation-related. (b) Example of a Subtask Stack.

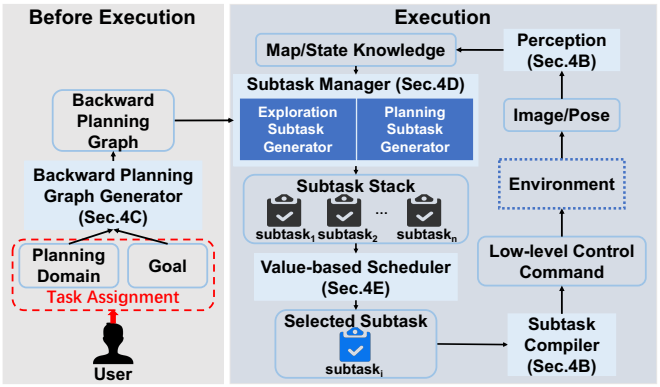


Fig. 3: An overview of our framework. Before execution, the Backward Planning Graph Generator generates a backward planning graph and feeds it to the Subtask Manager. During execution, the agent interacts with the environment through the Perception module and Subtask Compiler. The Subtask Manager and Value-based Scheduler are responsible for updating the Subtask Stack and selecting the optimal subtask based on current state.

$\texttt{par}(op) \subseteq \mathcal{V}$ is a list of parameters, $\texttt{pre}(op), \texttt{eff}^+(op)$, and $\texttt{eff}^-(op)$ are subsets of $\mathcal{P}(\texttt{par}(op))$ representing preconditions, effects, and delete effects respectively. The ground action $op(c)$ is obtained by instantiating op with real object entities $c = \langle c_1, ..., c_n \rangle$ in $\mathcal{C}$.

In this paper, we use PDDL, a de-facto standard input language of many symbolic planning systems, to model the planning problems. It separates the definition of a planning problem into two parts: the domain description $\mathcal{D}$ and the problem description $\mathcal{Q}$. $\mathcal{D}$ is a general model that captures the common parts among all problems, and $\mathcal{Q}$ is a specific problem instance based on this domain. $\mathcal{D}$ is a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{T}, \mathcal{V} \rangle$, defining predicates (templates for logical facts), operators, object types and parameters. $\mathcal{Q}$ is a tuple $\langle \mathcal{D}, \mathcal{C}, s_0, \mathcal{G} \rangle$ where $\mathcal{D}$ is the template domain, $\mathcal{C}$ is the set of object entities, $s_0 \subseteq \mathcal{P}(\mathcal{C})$ is the initial state, and $\mathcal{G}$ is the goal represented as a first order formula over $\mathcal{P}, \mathcal{V}$ and $\mathcal{C}$.

Task planning deals with synthesizing plans automatically that combine basic operators to achieve a high-level goal. A plan for a planning problem $\mathcal{Q}$ is an operator sequence $\langle op_1(c_1), ..., op_n(c_n) \rangle$, satisfying the existence of a state sequence $\langle s_1, ..., s_n \rangle$, such that $\texttt{pre}(op_i(c_i)) \subseteq s_{i-1}$, $s_i = s_{i-1} \cup \texttt{eff}^+(op_i(c_i)) \setminus \texttt{eff}^-(op_i(c_i))$ for every $1 \leq i < n$, and $s_n \models \mathcal{G}$. When applying symbolic planning to solve real-world robotics problems, each operator op is accompanied with a policy $\pi(op)$ that maps it to low-level control actions, and an occupancy grid map $\mathcal{M}$ of the scene is maintained for path planning and marking the positions of the robot and objects. Therefore, we denote the robot task planning problem in unknown environments as $\mathcal{Q}_u$, which is a tuple $\langle \mathcal{D}, \mathcal{C}, s_0, \mathcal{G}, \mathcal{M} \rangle$, where $\mathcal{C} = \emptyset, s_0 = \emptyset$, and all cells in $\mathcal{M}$ are in unknown states from the beginning, and the agent only knows $\mathcal{D}$ and $\mathcal{G}$. Besides, $\mathcal{G}$ is fully parameterized as $\mathcal{C} = \emptyset$ at first. The agent needs to explore the environment, extend the lists of entities and states, update the map step-by-step, and plan to achieve the goal.

In addition to the definitions of these basic concepts, we consider fine-grained classification of the predicates and operators in the planning domain (Figure 2(a)). According to current robotic ability knowledge base (RFUniverse [21], VirtualHome [22] and the H2020 Robotics MAR [23]), the robots' atomic actions are categorized into **navigation-related** and **manipulation-related**. Navigation-related operators are relevant to the robot's motion and only change the robot's state, whose effects are navigation-related predicates. Manipulation-related operators enable the robot to manipulate objects in the environment and change the state of the interacted object, whose effects are manipulation-related predicates. Besides, there is a class of predicates describing the specific property of objects, named as **property-related**, which is static during task execution.

IV. Methods

This section introduces the proposed framework and how to exploit it to co-optimize exploration and task planning.

A. Framework

Our framework (Figure 3) comprises five modules: Perception, Backward Planning Graph Generator, Subtask Manager, Value-based Scheduler, and Subtask Compiler. Before starting an episode, the Backward Planning Graph Generator creates a backward planning graph based on the user-assigned task. This graph is only generated once and is maintained and employed for planning subtask generation in Subtask Manager. During task execution, the Perception module extracts map and state knowledge needed for exploration and planning from raw sensor data. The two originally independent processes, i.e., robot exploration and symbolic planning, are decomposed into multiple subtasks with unified structures in the Subtask Manager, which generates and updates subtasks based on the current knowledge and the backward planning graph. The Value-based Scheduler chooses which subtask is optimal and should be executed to advance the current state towards the goal state. The Subtask Compiler compiles the selected subtask to a sequence of low-level control commands that the robot can execute in the environment. This process is repeated until the entire task has been accomplished.

This article has been accepted for publication in IEEE Robotics and Automation Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LRA.2022.3214784

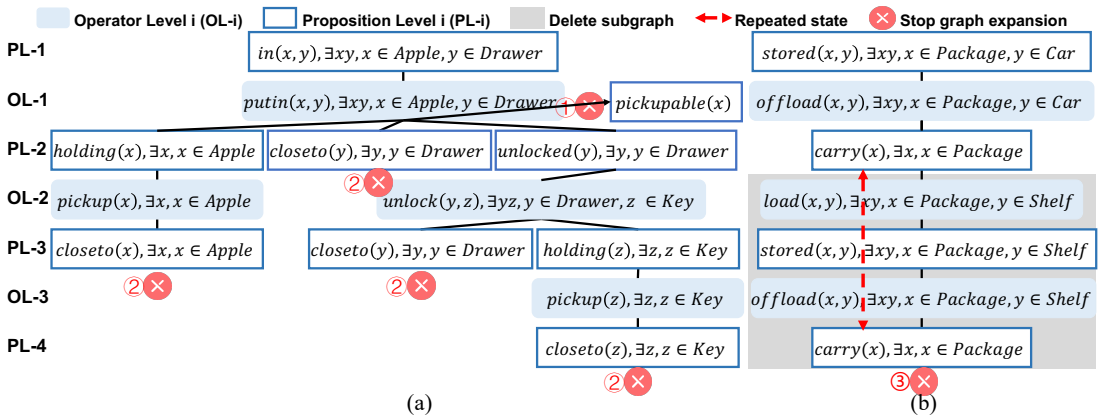


Fig. 4: (a) The backward planning graph for the task "put an apple in a drawer". (b) A part of the backward planning graph for the task "store a package on a car".

B. Perception & Subtask Compiler

The Perception module is responsible for building an occupancy map and transferring the sensor data obtained from the environment to the planning knowledge. Concretely, this module updates the set of object entities $\mathcal{C}$, the state information s containing facts, and occupancy grid map $\mathcal{M}$ at every step based on real-time sensor data. The Subtask Compiler receives the selected subtask and compiles its plan (a sequence of high-level symbolic operators) to the low-level control commands. For the navigation-related operators, the A* algorithm is used for path planning, while for the manipulation-related operators, APIs provided by the simulator are employed to interact with objects. We build these two modules upon the corresponding parts in OGAMUS [14], and show that the other three modules can improve the task efficiency significantly at the same level of perception and motion ability.

C. Backward Planning Graph Generator

The Backward Planning Graph Generator provides a backward planning graph after the user assigns a task. This graph is designed to find the intermediate states that the agent must experience before achieving the final goal, which are exploited in Section IV-D2 for subtask generation.

Unlike the Planning Graph [24], a Backward Planning Graph $\mathcal{BPG}$ is a directed and leveled graph with two types of nodes and two types of edges (Figure 4). The levels alternate between proposition levels containing proposition nodes (each labeled with a predicate) and operator levels containing operator nodes (each labeled with an operator). Edges in a $\mathcal{BPG}$ are established to represent relations between operators and propositions. For operator nodes in the operator-level i, "precondition-edges" connect them to their preconditions in proposition-level $i+1$, and "effect-edges" connect to their effects in proposition-level i.

The graph can be generated level-by-level, and for every proposition node in PL-i, we first search all operators in $\mathcal{O}$ and match their effects with the proposition node. If matched, i.e., the operator can assist the proposition, the operator is added into the OL-i. Then we generate PL-$(i+1)$ by adding the preconditions of all operators in OL-i to this level. The main difficulty in generating a $\mathcal{BPG}$ is defining the termination condition. The traditional planning graph generation [24], [25] starts with the initial global state, which is unavailable from the beginning in unknown environments, and ends when every goal proposition is present at a proposition level. On the contrary, a $\mathcal{BPG}$ generation starts with the given goal $\mathcal{G}$ and lacks termination conditions.

In order to bound the Backward Planning Graph extension, we design three termination rules (①②③ in Figure 4): ① The operator node does not add the property-related predicates in its preconditions to the proposition level. In Figure 4(a), `pickupable(x)` is a property-related proposition, so it will not be extended. ② The proposition node containing the navigation-related predicate does not extend. In Figure 4(a), all the `closeto(x)` nodes stop expansion. ③ The repeated proposition node stops expansion. Besides, the subgraph between the two same nodes is deleted. In Figure 4(b), the `carry(x)` node appears twice, so the latter one stops expansion and the part between the two nodes is deleted.

D. Subtask Manager

In the developed framework, a subtask is a structure that contains three attributes: a subgoal represented as a predicate, a plan to achieve the subgoal, and a value. For simplicity, we use the subgoal to name the subtask, with Figure 2(b) illustrating some subtask examples.

1) Exploration Subtask Generator: The exploration aims to obtain all the information about the unknown environment and build a map. Most exploration approaches are based on frontiers, where at each processing step, the agent moves to one frontier in the current map, and the exploration ends until there is no frontier. Inspired by these popular exploration methods, we design an algorithm to generate the *subtask* for exploration (Algorithm 1). The inputs of Algorithm 1 are the planning domain $\mathcal{D}$, current map $\mathcal{M}$, object entities $\mathcal{C}$, and state s, defined in Section III.

DetectFrontiers: First, the agent acquires the frontiers `frs` in the current map $\mathcal{M}$ (line 1). The method considers that the agent traverses each point in $\mathcal{M}$ and then selects the boundary points in the unknown region and the known free region. Then clustering methods [26] reduce the number of frontiers for computational efficiency.

Algorithm 1 Subtask generation for exploration

Require: $\mathcal{D}$, $\mathcal{M}$, $\mathcal{C}$, current state s
Ensure: a subtask list st_list_e
1: `frs` ← DETECTFRONTIERS($\mathcal{M}$)
2: Create an empty subtask list st_list_e
3: **for** each `fr` in `frs` **do**
4: Create an empty subtask st_e
5: $\mathcal{C}_e$,$\mathcal{M}_e$ ← ADDFRONTIER($\mathcal{C}$,$\mathcal{M}$,`fr`)
6: $st_e.subgoal$ ← CREATESUBGOAL($\mathcal{D}$,$\mathcal{C}_e$)
7: $st_e.plan$ ← PLAN($\mathcal{D}$, $\mathcal{C}_e$, s, $st_e.subgoal$)
8: $st_e.value$ ← COMPUTEVALUE($\mathcal{M}_e$, s, $st_e.plan$)
9: st_list_e ← APPENDSUBTASK(st_list_e, st_e)
10: **end for**
11: **return** st_list_e

Algorithm 2 Subtask generation for symbolic planning

Require: $\mathcal{D}$, $\mathcal{M}$, $\mathcal{C}$, current state s, $\mathcal{BPG}$
Ensure: a subtask list st_list_{sp}
1: $\mathcal{BPG}$ ← PRUNEGRAPH($\mathcal{BPG}$,s)
2: `subgoals` ← EXTRACTFROMBPG($\mathcal{BPG}$)
3: `subgoals` ← INSTANTIATE($\mathcal{C}$,`subgoals`)
4: Create an empty subtask list st_list_{sp}
5: **for** each `subgoal` in `subgoals` **do**
6: Create an empty subtask st_{sp}
7: $st_{sp}.subgoal$ ← `subgoal`
8: $st_{sp}.plan$ ← PLAN($\mathcal{D}$, $\mathcal{C}$, s, $st_{sp}.subgoal$)
9: $st_{sp}.value$ ← COMPUTEVALUE($\mathcal{M}$, s, $st_{sp}.plan$)
10: st_list_{sp} ← APPENDSUBTASK(st_list_{sp}, st_{sp})
11: **end for**
12: **return** st_list_{sp}

Then the generator creates a list containing all the exploration subtasks for each frontier `fr` in `frs` (lines 2-10).

AddFrontier: Since the frontiers change dynamically over time, we build a new and temporary entity list and map for every frontier `fr` by adding its entity and location into current $\mathcal{C}$ and $\mathcal{M}$ (line 5).

CreateSubgoal: The subgoal of st_e is obtained by instantiating the navigation-related predicate in $\mathcal{D}$ with the frontier in $\mathcal{C}_e$ (line 6). Considering the planning domain in Figure 2 for example, the navigation-related predicate in $\mathcal{D}$ is `closeto`, and the frontier in $\mathcal{C}_e$ is $\texttt{fr}_1$, so the subgoal is $\texttt{closeto}(\texttt{fr}_1)$.

Plan: A symbolic planner is invoked to solve the planning problem given the input planning domain $\mathcal{D}$, the updated set of objects, the current state s, and the exploration subgoal $st_e.subgoal$ (line 7). In particular, as $st_e.subgoal$ is a navigation-related proposition, $st_e.plan$ consists of a navigation-related operator.

ComputeValue: According to the plan, current map, and state, the value of the subtask can be estimated given a value function (line 8). Further details will be introduced in Section IV-E.

2) Planning Subtask Generator: Current methods can only combine exploration and symbolic planning in sequence because the symbolic planner only considers a single goal $\mathcal{G}$, and the planner can only output an empty operator sequence instead of a valid plan if the agent fails to obtain all necessary information during exploration to satisfy $\mathcal{G}$. For instance, if the goal is to put an apple on a table, the planner can only plan successfully if there is at least one object entity of type apple and one of type table in current $\mathcal{C}$ and state s.

However, we find that the agent must experience some intermediate states to achieve the final goal. For instance, the agent has to hold an apple before it achieves the goal of "having an apple on a table", while planning to achieve these intermediate states requires only part of the complete information. For instance, planning for $\texttt{holding}(\texttt{apple}_1)$ does not require the agent to know the existence and location of any table. This motivates us to decompose the original complete symbolic planning for the final goal into multiple subtasks with smaller subgoals.

The Backward Planning Graph in Section IV-C provides necessary intermediate states, and therefore we utilize it to generate subtasks for planning. Algorithm 2 illustrates the generation process in detail.

PruneGraph: At every step, the agent first prunes the graph to avoid generating subtasks that have been already completed (line 1). Concretely, the proposition nodes in the graph that can be satisfied by s are deleted, together with all the subgraphs that are extended from it. For the graph in Figure 4(a), if the agent picks up an apple and s contains $\texttt{holding}(\texttt{apple}_1)$, the three nodes ($holding(x)$, $pickup(x)$ and $closeto(x)$) and connected edges are pruned.

ExtractFromBPG: After graph pruning, the agent extracts subgoals, which are the essential intermediate states to achieve before finishing the goal (line 2). A straightforward approach is to consider all proposition nodes in $\mathcal{BPG}$ as subgoals. However, we find that not all the facts contribute to the final goal and are worth achieving in advance. For example, the navigation-related proposition in the agent's state constantly changes when the agent moves, so it should not be regarded as a valid subgoal. Therefore, we propose a method to extract valuable subgoals (Figure 4(a) as an example): 1) First we extract all the proposition nodes containing the manipulation-related predicate ($holding(x)$, $holding(z)$, $unlocked(y)$ and $in(x, y)$). 2) Then, we make a fine-grained selection among these propositions according to some rules, which can be easily defined and implemented through $\mathcal{BPG}$ and the domain description. For instance, $holding(z)$ node in PL-3 that is extended from $unlocked(y)$ in PL-2 has one neighborhood node $closeto(y)$, which is completely included in the neighborhood nodes of $unlocked(y)$. Therefore, $unlocked(y)$ node is not selected to avoid the robot first going close to the drawer to unlock it, then going close to the drawer again to put the apple. 3) At last, the remaining propositions ($holding(x)$, $holding(z)$, and $in(x, y)$) are regarded as valid subgoals.

Instantiate: In unknown environments, the goal formula G is parameterized as the agent has no prior knowledge about object entities in the scenario, and appears as a `forall` $\forall$ or `exists` $\exists$ statement. Therefore, the subgoals extracted from $\mathcal{BPG}$ are also parameterized. In order to consider as many situations as possible to find the optimal plan, the agent instantiate the subgoals with constants in $\mathcal{C}$ (line 3). For instance, $(holding(x), \exists x, x \in Apple)$ is a subgoal extracted

from $\mathcal{BPG}$ and there are two entities $\texttt{apple}_1$ and $\texttt{apple}_2$ in $\mathcal{C}$. The function transforms the parameterized subgoal to two instantiated subgoals $\texttt{holding}(\texttt{apple}_1)$ and $\texttt{holding}(\texttt{apple}_2)$.

Lines 7-9 are the same as lines 6-8 in Algorithm 1, where the agent creates the subgoal, plan, and value of the subtask and adds it to the subtask list. In particular, the plans of the subtasks in st_list_{sp} change over time. From the beginning, all plans are empty because the symbolic planner cannot find a plan for any subgoal. Nevertheless, with the discovery of more objects, more plans become valid.

After generating st_list_e and st_list_{sp}, the Subtask Manager merges them to obtain the Subtask Stack. Figure 2(b) depicts an example of a Subtask Stack.

E. Value-based Scheduler

We utilize the Subtask Stack to design a value-based scheduler deciding which subtask in the stack the agent should perform at each step. Completing the selected subtask aims to contribute as much as possible to achieve the final goal while requiring the minimum steps.

Two factors affect the realization of the final goal: 1) Information gained by exploring unknown areas. As mentioned above, the symbolic planner requires the necessary information to find a plan, with more information helping find the optimal plan. 2) Achievement of any intermediate subgoal. In Section IV-C, we highlight that to realize the final goal the agent needs to complete the subgoals in $\mathcal{BPG}$ in a bottom-up fashion. Therefore, achieving any subgoal means that the agent completes part of the task. Based on these factors, we propose a value function for each subtask to compute its value (line 7 in Algorithm 1 and line 9 in Algorithm 2) as follows:

$$V = -Cost_{plan} + \alpha Gain_{info} + \beta Gain_{task} \tag{1}$$

- $Cost_{plan}$ is the total number of steps to execute the subtask's plan. For instance, the cost of $\texttt{gocloseto}(\texttt{table}_1)$ is 10 if the agent requires ten movements to reach $\texttt{table}_1$, and the cost of $\texttt{pickup}(\texttt{apple}_1)$ is 1 as the `PickupObject` action takes one step in the simulator. The cost of a subtask without a valid plan is infinite.
- $Gain_{info}$ is the potential information gain, i.e., the volume of the unmapped area covered during subtask execution. It is widely used to select the next-best-view point to visit in autonomous exploration approaches [27], [28]. In our framework, for computational efficiency purposes, we estimate the $Gain_{info}$ of a subtask by placing the agent at the subtask destination and calculating the total number of the unknown cells inside the agent's Field of View (FOV) and visibility distance.
- $Gain_{task}$ corresponds to the reward gain according to the contribution a subtask makes to completing the final goal. The gain of each subtask is obtained using the backward planning graph $\mathcal{BPG}$. Concretely, we start from non-extended subtasks, such as $holding(x)$ and $holding(z)$ in Figure 4(a), and assign one to its $Gain_{task}$. The $Gain_{task}$ of the subtask on the lower proposition level is the sum of all $Gain_{task}$s of the subtasks that are extended from it. For example, the $Gain_{task}$ of $in(x, y)$ in Figure 4(a) equals to two. This summation design is heuristic because the lower the proposition level, the closer achieving the goal, and the more rewards should be given. Especially, $Gain_{task}$ of the subtask for exploration is zero.
- α and β are two parameters balancing the magnitude of the three values and adjusting the scheduling policy. Intuitively, if α is large, the subtask for exploration is prioritized, as exploration leads to more information gain. On the contrary, if β is large, the agent prefers to execute every subtask with a valid plan before continuing exploration. Therefore, appropriate values of α and β enable the agent to balance exploration and symbolic planning, combining their advantages better and improving task execution efficiency in unknown environments.

The designed value function affords each subtask in the subtask stack to compute the value of its plan, where the value-based scheduler is responsible for selecting the subtask with the highest value. For example, the scheduler selects subtask_3 (blue color) in Figure 2(b) as the current best subtask to execute because it has the highest value. By maximizing the proposed value function (robot exploration cares about $Cost_{plan}$ and $Gain_{info}$, and symbolic planning cares about $Cost_{plan}$ and $Gain_{task}$), we co-optimize robot exploration and task planning in a unified way.

V. Experiments

A. Simulator, Data Sets and Tasks

We experimentally evaluate our framework with the AI2-THOR [29] simulator, an open-source interactive photo-realistic environment for Embodied AI. The robot in AI2-THOR is equipped with an RGB-D camera and perceives the environment with a given FOV (set to 80°). As introduced in Section IV, our framework is designed to improve the robot planning ability in unknown environments without leveraging the perception ability. Thus we adopt a ground-truth Perception module in simulation for our framework and all baselines to achieve a fair comparison. AI2-THOR divides its operator APIs into two categories, navigation and manipulation, which is the same as our classification presented in Section III. The basic behaviors of the agent include moving forward of a given distance (set to 20cm), turning left or right of a given angle (set to 45°), picking up objects, putting an object onto or into the target receptacle, and opening or closing objects. By using these action APIs, we define the operators $\mathcal{O}$ in the planning domain $\mathcal{D}$ and implement these operators during simulation.

The following experiments involve three types of tasks, and the corresponding goals are:

1. Task On: Put an object of type t_1 onto an object of type t_2: the agent is required to find at least two objects of types t_1 and t_2 and put the one of type t_1 on top of the other of type t_2. For instance, the agent has to put an apple on a table. The corresponding goal is: $on(x, y), \exists xy, x \in Apple, y \in Table$.
2. Task In: Put an object of type t_1 into an object of type t_2 which is locked from the beginning: the agent has to find at least two objects of types t_1 and t_2 and one key, unlock the one of type t_2, and put the one of type t_1 into

TABLE I: Average episode length in different scenarios for three complex tasks.

	On						In						Cook					
	small		medium		large		small		medium		large		small		medium		large	
	single	multi	single	multi	single	multi	single	multi	single	multi	single	multi	single	multi	single	multi	single	multi
OGAMUS [14]	40.1	32.5	63.5	44.5	116.0	96.1	55.0	52.5	75.6	64.6	136.3	126.8	65.4	53.4	112.9	86.8	178.2	155.0
α=0.1, β=10	29.5	26.5	48.5	38.8	86.5	81.4	45.4	43.5	58.4	53.9	113.6	103.1	55.3	37.3	83.6	77.2	146.1	127.4
α=0.1, β=20	27.5	**22.0**	44.1	**34.9**	78.1	**72.2**	44.2	**38.6**	56.5	**47.2**	102.3	91.4	53.8	40.1	77.4	72.5	133.4	115.2
α=0.1, β=40	**27.2**	24.3	**43.3**	35.1	**77.6**	72.6	**43.9**	43.0	**55.4**	48.5	**99.0**	92.3	**51.9**	38.8	**72.3**	68.0	**128.6**	116.3
α=0.15, β=10	30.7	26.4	50.9	40.2	87.3	83.7	47.1	44.1	61.8	54.9	110.9	109.8	57.5	40.3	86.2	77.8	138.5	137.2
α=0.15, β=20	27.6	23.1	46.4	35.3	80.1	76.8	44.3	43.0	55.7	52.9	104.8	97.7	54.3	38.6	80.4	75.3	134.7	123.9
α=0.15, β=40	**27.2**	24.9	**43.3**	35.1	77.8	72.5	**43.9**	43.3	55.5	49.9	99.1	**90.9**	52.0	**36.9**	74.7	**67.6**	128.8	**114.9**
α=0.2, β=10	34.4	26.1	58.7	43.8	89.8	90.3	48.0	47.0	63.0	59.3	120.4	118.9	59.1	45.2	98.3	81.0	162.8	146.3
α=0.2, β=20	30.3	24.1	48.6	38.6	85.9	79.9	45.5	42.3	58.4	54.7	112.5	102.2	55.7	37.3	84.3	76.8	144.9	126.6
α=0.2, β=40	27.3	24.2	43.9	35.2	78.0	72.3	44.1	41.8	56.3	48.0	101.1	92.4	53.6	38.0	76.8	71.2	132.3	115.0

the other of type t_2. For instance, the agent has to put an apple into a locked drawer. The corresponding goal is: $in(x, y), \exists xy, x \in Apple, y \in Drawer$.

3. Task $Cook$: This task is that a home service robot prepares breakfast for users, including a fried egg and a piece of toasted bread. The agent has to find at least one egg and one piece of bread as ingredients and to find a pan and put it on a stove burner to fry the egg. To toast the bread, a toaster should be discovered and used. The corresponding goal is:$fried(x) \wedge toasted(y), \exists xy, x \in Egg, y \in Bread$.

An episode is obtained by randomly placing the agent in an unseen scene and providing it with a randomly generated goal. An episode's length is the number of interactions with the environment (open, move ahead, pick up each count as one action). We command the robot to finish the three tasks on scenes with different map sizes and numbers of goal-related objects. We use *kitchens* as *small* scenes ($\approx 15m^2$), *living rooms* as *medium* scenes ($\approx 30m^2$), and *apartments* as *large* scenes ($\approx 40m^2$). A *single* scene contains only one object per type, and a *multi* scene contains two objects per type. We run our algorithm with 100 episodes for each type of simulation scenario and collect the episode lengths when the agent achieves the assigned goal. The average episode length is used as an evaluation metric, and a shorter episode length means higher efficiency. Real-world experiments are conducted on an Ackermann mobile robot with an RGB-D camera for tasks *On* and *In*. The mapping and localization of this robot are achieved by deploying a visual SLAM system on it. For simplicity of implementation, we assume that the robot can manipulate an object virtually within a distance of 0.5m from it, instead of using a real robotic arm.

B. Parameter Selection

According to Section IV-E, the values of α and β have a significant influence on our framework's performance. Since we comprehensively consider the effects of the three terms from Equation (1), α and β cannot be too large to dominate V or too small to be neglected. Therefore, we determine a reasonable range value for α by randomly selecting simulation scenarios for the agent to explore, count the average number of steps ($Cost_{plan}$) required to finish an exploration subtask while considering the corresponding information gain ($Gain_{info}$), and estimate the range of α that balances these two values. The average value of α is 0.136, so we select $\alpha = 0.1$, $\alpha = 0.15$ and $\alpha = 0.2$ to study its impact. For β, we count the average number of steps to complete a random planning task in a known environment and given the related rewards ($Gain_{task}$). Based on the average value 24.8, we evaluate $\beta = 10$, $\beta = 20$ and $\beta = 40$.

C. Results

We challenge the suggested methods with different α and β values against OGAMUS [14], with the corresponding results reported in Table I. For tasks On, In, and $Cook$, our framework improves the task execution efficiency by 29.0%, 25.9% and 27.2%, respectively, due to: 1) The Subtask Manager and Value-based Scheduler enable the agent to execute the symbolic operators during exploration. Thus, the agent saves many steps needed initially to reach the repeated states. 2) The designed value function considers various factors, making the selected exploration subtasks more beneficial to task planning for the final goal. On the contrary, the robot exploration in OGAMUS is random and not linked to planning.

In the experiments, we study the influence of map size on performance. Since the same transition between two symbolic states requires more steps in large scenes, we find that our framework can save more steps on the large scenes than on the small ones for the same task. This indicates that our framework has greater utility in large scenes, e.g., factories and outdoors.

The experimental results show that different values of parameters α and β lead to different efficiency improvements. In general, if these parameters are within a reasonable range, the policy mainly depends on the ratio of β and α. When $\frac{\beta}{\alpha}$ is large, the value of the planning subtask is higher than exploration, and the agent prioritizes all encountered planning subtasks. In *single* scenarios, whenever the agent finds a goal-related object, it does not need to further explore to find another one, so a larger $\frac{\beta}{\alpha}$ often results in better performance. In particular, when the ratio is beyond a threshold, such as $\alpha = 0.15, \beta = 40$ and $\alpha = 0.1, \beta = 40$, the average episode length remains almost constant, as the value of any planning subtask is always higher than that of the exploration subtask. However, a larger $\frac{\beta}{\alpha}$ is not necessarily better in *multi* scenarios. In such scenes, additional exploration to discover more objects of the same type is helpful in finding the optimal plan. Therefore, the policy with $\alpha = 0.1, \beta = 20$ performs best in some *multi* scenarios.

In the real-world experiments, compared with OGAMUS, our method shortens the average trajectory length by 26.8% and 26.0% respectively. These results show that our framework can perform well without access to perfect world state and has good transferability from simulation to reality.

To summarize, our framework achieves 25%-29% efficiency improvement on three different tasks and shows greater potential in the large scenes. A larger $\frac{\beta}{\alpha}$ makes task execution more efficient when there is only one plan to achieve the goal. However, for planning problems with multiple feasible plans, the upper threshold of $\frac{\beta}{\alpha}$ should be appropriately set to avoid finding a plan with inferior performance. The most time-consuming module in our framework is Backward Planning Graph Generator, but it only works once before performing the task and has no influence on the online task execution.

VI. Conclusion & Future work

This paper proposes a framework for accomplishing complex tasks in unknown environments by combining robot exploration and task planning. We design a structure named subtask and suggest the corresponding generation algorithms to represent the two independent processes in a unified manner. Moreover, we design a value function and a scheduler, which consider the information gain, direct contribution to the goal, and execution cost, to co-optimize information acquisition and symbolic planning. Evaluation results on complex tasks show that our framework improves execution efficiency by 29% and has better potential in large scenes.

Future work will address the problem that the parameters α and β need to be tuned manually in scenes of different sizes by designing an adaptive module to update parameters online. Moreover, as different task definitions and PDDL descriptions affect the extraction of subgoals, we plan to test our framework on more tasks to enrich the fine-grained selection rules.

References

[1] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6793–6800.

[2] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 5588–5595.

[3] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[4] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[5] R. Dearden and C. Burbridge, "An approach for efficient planning of robotic manipulation tasks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 23, 2013, pp. 55–63.

[6] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 2970–2977.

[7] Z. Ma, Y. Zhuang, P. Weng, H. H. Zhuo, D. Li, W. Liu, and J. Hao, "Learning symbolic rules for interpretable deep reinforcement learning," *arXiv preprint arXiv:2103.08228*, 2021.

[8] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," *arXiv preprint arXiv:1609.05518*, 2016.

[9] A. d. Garcez, A. R. R. Dutra, and E. Alonso, "Towards symbolic reinforcement learning with common sense," *arXiv preprint arXiv:1804.08597*, 2018.

[10] Y. Xu, J. Yu, J. Tang, J. Qiu, J. Wang, Y. Shen, Y. Wang, and H. Yang, "Explore-bench: Data sets, metrics and evaluations for frontier-based and deep-reinforcement-learning-based autonomous exploration," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 6225–6231.

[11] J. Hoffmann, "Ff: The fast-forward planning system," *AI magazine*, vol. 22, no. 3, pp. 57–57, 2001.

[12] D. Gordon, D. Fox, and A. Farhadi, "What should i do now? marrying reinforcement learning and symbolic planning," *arXiv preprint arXiv:1901.01492*, 2019.

[13] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators," in *Conference on Robot Learning*. PMLR, 2020, pp. 603–616.

[14] L. Lamanna, L. Serafini, A. Saetti, A. Gerevini, and P. Traverso, "Online Grounding of Symbolic Planning Domains in Unknown Environments," in *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, 2022, pp. 511–521.

[15] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox, "Transferable task execution from pixels through deep planning domain learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 459–10 465.

[16] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015, pp. 333–341.

[17] F. Martín, J. G. Clavero, V. Matellán, and F. J. Rodríguez, "Plansys2: A planning system framework for ros2," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9742–9749.

[18] M. Levihn, L. P. Kaelbling, T. Lozano-Pérez, and M. Stilman, "Foresight and reconsideration in hierarchical planning and execution," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 224–231.

[19] V. Vasilopoulos, W. Vega-Brown, O. Arslan, N. Roy, and D. E. Koditschek, "Sensor-based reactive symbolic planning in partially known environments," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5683–5690.

[20] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez, "Representation, learning, and planning algorithms for geometric task and motion planning," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 210–231, 2022.

[21] H. Fu, W. Xu, H. Xue, H. Yang, R. Ye, Y. Huang, Z. Xue, Y. Wang, and C. Lu, "Rfuniverse: A physics-based action-centric interactive environment for everyday household tasks," *arXiv preprint arXiv:2202.00199*, 2022.

[22] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[23] "Robotics 2020 multi-annual roadmap for robotics in europe," 2020. [Online]. Available: www.eu-robotics.net/sparc/upload/Newsroom/Press/2016/files/H2020_Robotics_Multi-Annual_Roadmap_ICT-2017B.pdf

[24] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[25] D. Bryce and S. Kambhampati, "A tutorial on planning graph based reachability heuristics," *AI Magazine*, vol. 28, no. 1, pp. 47–47, 2007.

[26] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[27] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon" next-best-view" planner for 3d exploration," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1462–1468.

[28] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-d environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[29] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," *arXiv preprint arXiv:1712.05474*, 2017.