Task and Motion Planning for Execution in the Real

Tianyang Pan, Rahul Shome, Lydia E. Kavraki

***Abstract*—Task and motion planning represents a powerful set of hybrid planning methods that combine reasoning over discrete task domains and continuous motion generation. Traditional reasoning necessitates task domain models and enough information to ground actions to motion planning queries. Gaps in this knowledge often arise from sources like occlusion or imprecise modeling. This work generates task and motion plans that include actions cannot be fully grounded at planning time. During execution, such an action is handled by a provided human-designed or learned closed-loop behavior. Execution combines offline planned motions and online behaviors till reaching the task goal. Failures of behaviors are fed back as constraints to find new plans. Forty real-robot trials and motivating demonstrations are performed to evaluate the proposed framework and compare against state-of-the-art. Results show faster execution time, less number of actions, and more success in problems where diverse gaps arise. The experiment data is shared for researchers to simulate these settings. The work shows promise in expanding the applicable class of realistic partially grounded problems that robots can address.**

***Index Terms*—Task and Motion Planning. Robust Execution.**

I. Introduction

Autonomous robots that operate in realistic environments and accomplish complex task objectives need to be capable of deciding sequences of actions to take and corresponding motions to execute. Task and Motion Planning (TAMP) [1], [2], [3] is a popular class of techniques that is applied to such problems. Given a symbolic representation of a task objective, task planning produces an action sequence called a task plan. A grounding process is necessary to generate motion planning queries for each discrete action. A motion planner can report queries to be infeasible due to geometric constraints such as collisions. A typical TAMP planner [1] will feed back such failures as symbolic constraints to compute an alternative task plan, until a valid task and motion plan is discovered. While being a general and effective paradigm, a prerequisite for this strategy is the existence of knowledge about the symbolic and geometric representations of the world that is sufficient to ground a symbolic action to a motion planning query. This information is typically expected from sensing or part of modeling. This work focuses on task and motion planning problems where not enough information is available to complete every grounding step at planning time.

A variety of realistic situations introduce partial knowledge of the world for a given task, violating the assumptions of the state-of-the-art TAMP solvers. For example, as in Fig. 1, we may know that two objects are on the table and one object is in the drawer, but we do not know the geometric poses of some of the objects due to occlusions, inaccurate sensors, etc. The object pose is necessary to generate robot grasping configurations and motions that achieve there. Traditional TAMP approaches struggle to address problems that might need to ground a `Pick` action that needs to interact with an unknown object. Therefore, a discrete plan from the task planner may consist of actions that can be fully grounded to continuous motions through motion planning, as well as actions that *cannot* be grounded due to lack of enough knowledge of the world. Partial grounding creates *gaps* in the resulting task and motion plan. The gaps may arise due to various reasons including perception uncertainties, imprecise simulations, modeling gaps, etc., as illustrated in Fig. 2. We observe that actions that could not be grounded at planning time might be necessary for the task and could be reasoned during execution time. This work focuses on how to address scenarios where only partial grounding is possible, and proposes to extend TAMP with closed-loop behaviors that are deployed during execution.

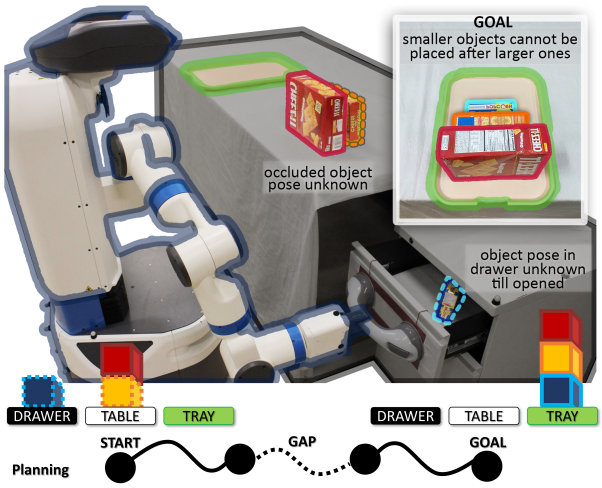

Fig. 1. The task is to move three objects to the goal tray. The full states of the dotted objects are not known precisely. A task and motion plan will have gaps which can only be filled in during execution, e.g., the blue object's pose can only be known after opening the drawer. The problem imposes constraints visualized in a block-world-like diagram at the lower part of the figure.

Recent advances [4] focus on TAMP with sensing uncertainties, and have proposed a framework that integrates a TAMP solver more closely into the typical Sense-Plan-Act loop, leveraging sensing and execution to gain more information. The motion planning difficulty is deferred till execution. Recent work [5] tackles TAMP problems where underlying models of the problem are incomplete, by repeatedly executing one or more steps of the current task and motion plan, gathering new information, re-formulating a new TAMP problem, and replanning. Other types of gaps such as stochas-

The authors are with the Department of Computer Science, Rice University; (tianyang.pan@rice.edu, kavraki@rice.edu). RS is currently with the School of Computing, the Australian National University (rahul.shome@anu.edu.au). The work was supported in part by NSF 1830549, NSF CCF-2336612 and Rice University Funds.

Fig. 2. The source of gaps can be diverse, including (from left to right) occlusions, imprecise simulations, and object modeling gaps (two paper bags look the same but contain different groceries).

tic execution outcomes are not considered. The sources of gaps are diverse and might only be known for parts of the TAMP domain. We focus on such real-world problems in which we are aware that some types of knowledge gaps may exist. *This propels our study into maintaining the benefits of coupled task and motion reasoning, while addressing incomplete* TAMP *domains with gaps that can be filled in during execution.*

We note that TAMP solvers are very powerful for problems where enough information is at hand. In motivating scenarios like Fig. 1, most of the problem might be known and possible to ground (e.g., the position of the table and the robot). We propose to build a framework that can leverage existing TAMP solvers for the part of the task where we have good enough information, and leave gaps in the plan for actions that cannot be fully grounded during planning (Fig. 1). The output of the planning phase is a partially grounded task and motion plan with gaps. This is sent to the execution layer, where we leverage provided closed-loop *behaviors*, as shown in Fig. 3. These behaviors can be human-designed modules, learning-based local policies, etc., that can be triggered during execution to ground individual actions where gaps exist. Even if a behavior is carefully designed, its execution may still fail. Our framework allows feedback from such failures as execution-layer constraints. Overall, the framework proceeds in an incremental fashion computing task plans till a partially grounded sequence of feasible motions plans and behaviors is generated. Behavior failures during execution trigger generation of alternative task plans, till the robot successfully accomplishes the task in execution using some combination of precomputed motion plans and behaviors.

The primary contribution of the current work is to design a general and extensible framework for **Task and Motion Planning for Execution in the Real (TAMPER)** that can address (a) dealing with symbolic states and actions which might only be partially grounded (creating *gaps*), (b) bridging the *gaps* using closed-loop behaviors comprising human-designed steps or learned controllers, and (c) recovering task-level constraints from the execution feedback from behaviors. We evaluate our proposed framework extensively on *real-world benchmarks* to demonstrate the efficacy of combining model-based TAMP reasoning with closed-loop behaviors. Our method succeeds more often, and takes fewer actions in real world settings with partial groundings, compared against a baseline implemented based on [5]. We also share a dataset showcasing real-robot TAMP problems that include problem definitions, recorded online data, and instructions to replicate experimental settings.

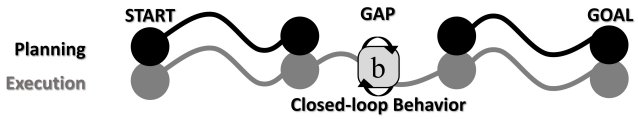

Fig. 3. This work introduces behaviors to bridge gaps in the task and motion plan during execution time, and recovers constraints from the task, motion, and execution domains.

II. Related Work

A. Task and Motion Planning

There are a variety of approaches proposed for integrated TAMP [6]. A typical approach [1] is to handle task planning using a general-purpose satisfiability modulo theories (SMT) solver [7] that supports incremental solving to efficiently consider motion constraints as feedback from motion grounding failures. Some works leverage black-box samplers encoded as streams to efficiently compute feasible plans [2]. TAMP has also been formulated as an optimization problem in [3]. Multi-modal motion planning (MMMP) methods represent the underlying discrete structure of TAMP problems as mode families, and compute the mode transitions as well as the single-mode motions [8], [9], [10]. The above methods usually assume perfect sensing and deterministic execution outcomes, and focus on planning with complete knowledge of the world. If these assumptions do not hold, such strategies exhaust all possible discrete plans without finding a feasible one.

TAMP in stochastic domains is gaining more and more interest recently. Some works formulate the stochastic TAMP problem as a Markov Decision Process (MDP) [11], [12], assuming a known transition model. In [13], the problem of TAMP with failing executions is modeled as an MDP with unknown transition probabilities, where a Beta-Binomial model is leveraged to maintain the belief of the unknown probabilities to minimize the expected number of actions executed to finish tasks. These works focus on stochastic execution outcomes and assume perfect sensing.

There are approaches that address underlying uncertainty by formulating partially-observable TAMP domains and planning in a belief space. The general formalism of Partially Observable Markov Decision Processes (POMDPs) [14] provides a way to compute optimal policies, but is typically computationally expensive. Efficient online, approximate variants [15], [16] have been proposed. Despite this, careful modeling of state and action spaces is necessary to both reasonably model the problem and maintain reasonable performance. The fully defined robotic task and motion domain presents tractability issues since the dimensionality of the belief space grows with the cardinality (or size) of the state space.

There have been works that leverage or extend these online POMDP solvers and apply them to robotics manipulation domains [17], [18]. They focus on reasoning over discrete actions choices, while computing continuous motions for high-DoF manipulators is not addressed, in contrast to TAMP literature whose forte is the coupled discrete-continuous problem. In order to tackle TAMP problems with uncertainty, different representations and approximations have been proposed for the belief over the partially observable object poses or robot

joint values, including Kalman filtering, particle filtering, or Nonparametric Belief Propagation [4], [19], [20]. These are mostly closed-loop TAMP methods with the focus on policy computation that needs to react on the fly to new observations with bounded horizon reasoning. Partially observable TAMP has also been integrated with trajectory-optimization-based techniques [21]. Strong assumptions are often necessary to simplify problem modeling [4], [19], [20], [21], which allows the resulting POMDP to be solved more efficiently. For example, the probabilistic model of transitions can be determinized to compute plans in a tractable way [4], [19]. Moreover, all the above works typically focus on a single type of gap at a time in terms of the observation model, for instance occlusion or noise in object pose estimation [4], [19], [20], [17], [18], or partially observable attributes of objects [21] (e.g., the color of the opposite side of a cube cannot be seen).

It is important to point out that many types of partial groundings (gaps) could exist in TAMP problems. This again raises open problems in modeling and scalability. Moreover, it remains to be explored, how long-horizon task-level logical constraints can be enforced in manipulation tasks using a POMDP model, and it is unclear how to represent and use geometric constraints coming from continuous motion computation and execution within in a POMDP framework. Notably, TAMP on its own cannot address many of these complexities, and the proposed work pushes the traditional constraint-based TAMP strategy that handles long-horizon, discrete-continuous, high-dimensional search spaces into more realistic partially-grounded scenarios. We address such a new class of problems by using provided behaviors that deal with various types of local gaps to efficiently reason about planning and execution.

A recent related work [5] plans with an incomplete model of the world (e.g., unmodeled objects and occlusion). It proposes to estimate and reason over the properties and affordances of the objects. It keeps replanning with the detected objects in the now until it reaches the goal or a dead end. Our work can deal with not only gaps coming from perception, but also other general types of gaps (e.g., stochastic execution outcomes), as long as specialized execution-level behaviors are provided. Our main contribution is a general formulation and framework for TAMP with partial groundings, while the necessary engineering effort for each particular problem is left modularized and flexible as input behaviors.

B. Reactive Manipulation

Substantial research effort has been devoted to developing methods that are reactive and adaptive to external interventions in real-world robotic tasks. Reactive synthesis approaches [22], [12] aim at fully computing a policy over the abstraction represented by some formal language such as Linear Temporal Logic (LTL) [23] to handle human intervention or environmental stochasticity during execution time. Although these methods have been used for high-DoF robot manipulation tasks, searching for the abstract policy is already highly complex; so they usually simplify the computation of low-level motions. Synthesizing such a policy requires full knowledge of the task and motion domain, and when the domain changes the policy must be recomputed. To more efficiently react to environmental model changes, another approach partially constructs the search graph of the task domain and exploits the modularity of Behavior Trees (BTs) [24] as the execution layer to manage the given low-level control policies [25]. In [26], TAMP is formulated as an optimization problem over relative object poses, and reactive controllers can be leveraged to adapt to real-world changes. It is assumed that the controllers can always react to the disturbances during execution. Hence recomputing a new task plan is not necessary. All the above synthesis methods mostly compute a policy for the full task given, which becomes computationally prohibitive when considering execution-level constraints or solving long-horizon tasks. We focus on problems where, despite the problem not being fully grounded during planning, we can efficiently reason over motion constraints and execution failures.

Plan execution monitoring methods [27] specifically focus on detecting the difference between world states and expected states, and recovering from the discrepancies. There have been works that consider execution failures by diagnosing broken parts of the robotic system with either fully observable [28] or partially observable state spaces [29]. Another work reacts to temporal or resource constraints that emerge at execution time [30]. These methods usually do not focus on the combined task and motion planning problem for high-DoF robots, and mostly only consider cases where the *gap* of grounding comes from imperfect simulation model (malfunctioning robot or unmodeled physics) of the real world. In this work, we focus on the general problem of partial groundings in TAMP, where diverse types of *gaps* can arise.

Another line of research leverages suitable execution-level architectures to achieve reactive and robust behaviors. BTs have become popular as a control structure for complex robot systems because they allow modularity, hierarchy, and feedback control [24], [25], [31]. Back chaining from the goal condition for planning has been used to create and update BTs to react to external disturbances [32]. Another work proposes Robust Logical-Dynamical Systems (RLDS) as an alternate structure that has equivalent expressiveness as BTs and algorithms to compose and execute an RLDS [33]. These works usually defer motion planning to execution time. In constrast, we leverage the power of TAMP planners that can discover some motion or geometric constraints (e.g., infeasible motions from certain states) at the initial planning stage, before even entering the expensive real-world execution stage. As pointed out in Sec. I, we consider cases where even the input behaviors are not guaranteed to complete an action, making feedback of constraints from execution layer and replanning inevitable and necessary — something that is typically not addressed in many of the above methods.

In another work [34], active inference approaches are integrated with BTs to enhance the robustness of the given nominal plan (in the form of BT) to unexpected disturbances and partial observability. The primary focus is to enhance the given offline nominal plan during runtime, while our approach can efficiently leverage such behaviors with the proposed task and motion planning and execution framework.

Many of the works in this section are complementary to

our proposed framework. TAMPER can efficiently reason over long-horizon task and motion planning problems where partial groundings exist due to lack of full knowledge of the real-world domain. We leverage provided behaviors (see definition in Sec III-B), which can be in the form of these powerful online local recovery policies, to ground individual actions where grounding gaps exist.

C. Learning-based Methods

Machine learning techniques have been leveraged to learn abstract hierarchical policies for various robotic tasks from human-designed reward functions or expert demonstrations [35], in the form of Finite State Machines or BTs [36]. However, they usually consider motion planning as an online module that does not fail. Since the completeness of planning for high-DoF robotic arms in complex scenes cannot be guaranteed, we inherently reason over motion planning failures as constraints in line with other typical TAMP methods.

Some learning-based methods aim at learning the grounding itself, applied to recover symbols that permit planning using skills [37] or LTL expressions [38]. Our focus is on problems where grounding functions might exist but cannot be applied due to uncertainty or insufficient information.

Some works combine Reinforcement Learning with TAMP planners to adapt to unseen environmental dynamics with mobile robots [39], or use unsupervised learning to collect data and predict feasibility of task plans in domains where execution may fail [40]. To address problems where the mapping between symbolic abstraction and real-world sensor data is hard to define, the approach in [41] learns both the high-level model to predict the symbolic state, and the low-level control policies for execution. Some work looks into the problem of aquiring neuro-symbolic skills given symbolic predicates through learning from demonstration [42]. To help with gaps in grounding unseen objects, another work aims at learning object-centric representations from RGB images for specific manipulation tasks, where the execution is composed with given primitive skills [43]. There also exist several deep Reinforcement Learning approaches that let the robot learn skills for manipulation [44] like screwing the cap of a bottle [45], etc. Acquiring such skills is not the focus of this paper. Our framework can readily Learning-based methods are hard to scale to general longer-horizon manipulation tasks that TAMP planners excel at solving [44]. Also, simulation data is necessary because acquiring a large amount of real-world data for robotic manipulation tasks is difficult and expensive. But a simulator might not be a perfect model of the real world, presenting challenges in applying the policies learned in simulation to real robots, known as the sim-to-real gap [46]. Some methods address this problem, but mostly only focus on recovery from failures caused by unmodeled dynamics, physical interactions, or system errors [13], [29].

Again, many of the powerful learning-based local policies in this subsection can be leveraged as input behaviors for TAMPER to ground individual actions where grounding gaps exist.

III. Problem Formulation

We introduce the typical formulation of TAMP in sec. III-A, and then extend the formulation to the problem we aim to solve in sec. III-B.

A. Task and Motion Planning

Task Planning: Task planning is defined over a domain D_{TP} that consists of

- a finite set of states $\mathcal{S}$, where each state $s \in \mathcal{S}$ is defined over a set of boolean state variables $\mathcal{P} = \{p_0, p_1, ..., p_n\}$, i.e., $s \in 2^{\mathcal{P}}$.
- a finite set of actions $\mathcal{A}$ that allow transition from one state that satisfies the action's pre-conditions (denoted as $\texttt{pre}(a) \subseteq \mathcal{S}$) to another state that satisfies the action's end-effect (denoted as $\texttt{eff}(a) \subseteq \mathcal{S}$).
- a finite set of constraints Φ, where each constraint ϕ is a logical assertion indicating an action cannot be taken from a set of states, i.e., $\{s_1, s_2, ...\} \Rightarrow \neg a$. Please refer to [1] for details.
- an initial state $s_{init} \in \mathcal{S}$,
- and a finite set of goal states $\mathcal{S}_{goal} \subseteq \mathcal{S}$.

The solution to a task planning problem is a task plan T which is a sequence of actions $a_0, a_1, ..., a_l$. Each action a_i transitions from its state $s_i \in \texttt{pre}(a_i)$ to $s_{i+1} \in \texttt{eff}(a_i)$, where $s_0 = s_{init}$ and $s_{l+1} \in \mathcal{S}_{goal}$. The task plan must also satisfy all the constraints. Formally, $\forall i, \forall \phi \in \Phi, a_i \vDash \phi$.

Motion Planning: The motion domain D_{MP} describes a d degree of freedom (DoF) robot defining a configuration space $\mathcal{X}$, a subset of which is valid $\mathcal{X}_{\text{free}} \subseteq \mathcal{X} \subset \mathbb{R}^d$. A trajectory is a time parameterized curve $\pi : [0, 1] \rightarrow \mathcal{X}_{\text{free}}$, which is called a solution to a motion planning problem between a start $x_0 \in \mathcal{X}$ and a goal $x_1 \in \mathcal{X}$ if $\pi(0) = x_0, \pi(1) = x_1$. It is typically assumed that it suffices to find a geometric path π_i for the robot to ground an action a_i into robotic motions. The validity of these motions, the corresponding $\mathcal{X}_{free}$ depend on the geometric components of the workspace like static obstacles and poses of movable objects.

Task and Motion Planning (TAMP): An instance of a TAMP problem includes an initial state s_0 and a set of goal states $\mathcal{S}_{goal}$. TAMP is the problem of finding a feasible n-step task and motion plan $\mathbf{T} = (\langle a_0, \pi_0 \rangle, ..., \langle a_n, \pi_n \rangle)$, where $a_0, ..., a_n$ is the task plan that satisfies the task planning domain and successfully transitions to a goal state $s_n \in \mathcal{S}_{goal}$. Each discrete action a_i transitions from state s_i to the result state s_{i+1} and corresponds to a feasible motion plan π_i.

Notably, the geometric feasibility depends on not only the static geometric parts of the environment but also task-dependent aspects like object or environment interactions or task-specific properties or constraints.

Execution after Planning: Once a feasible task and motion plan is computed, in open-loop execution the motion plan can then be sent to controllers which move the robot in the real world. This executes the computed plan.

Such open-loop execution succeeds on the real robot when the computation and planning used models of the world that closely matches the real world and the robot controller is accurate. A successful open-loop execution is expected to

show little deviating from the computation of the planner, as illustrated in Fig. 4a.

The focus is shifted to scenarios where the computation and open-loop execution of task and motion plans can prove insufficient to successfully complete the task during robot execution. In contrast to open-loop execution, interleaving planning and execution alongside combined reasoning across both presents a more powerful paradigm. *We formulate the problem where both planning and execution for task and motion goals are under consideration.*

We are interested in TAMP problems where we do not have enough information of the world, or in other words, the assumption of a perfect model of the world does not hold. In such cases, some of the symbolic states and actions may remain partially grounded, and real-world execution is necessary to obtain more information to fully ground and execute such actions.

B. TAMP for Execution

In this section we define what partial grounding means in our context, and proceed to expand on the formulation of TAMP to involve executions.

World State Space: A world state is a vector of all the attributes that models the world inside planning. This goes beyond the degrees of freedom of the robot configuration and can include other properties of the world relevant to the problem. The world state space $\mathcal{W}$ can include different continuous or discrete state spaces $\mathcal{W} = \mathcal{X}_{\text{robot}} \times \mathcal{X}_{\text{object}_1} \times ... \times \mathcal{X}_{\text{property}_1} \times ...$, where $\mathcal{X}_{\text{robot}}$ is the configuration space of the robot, $\mathcal{X}_{\text{object}_1}$ typically describes the SE(3) pose of an object, and $\mathcal{X}_{\text{property}_1}$ can describe a property of the world that is relevant to the task (e.g., whether the table is clean or dirty, or whether the drawer is open or closed).

Grounding Symbolic States and Actions: Grounding is the value assignment of all the symbolic variables in an expression. In the context of TAMP, each symbolic state s can be fully grounded to a set of attributes describing the world state. This grounding operation can create a one-to-many mapping to a set of (usually infinitely many) possible world states, $\mathbf{G}(s_i) \subset \mathcal{W}$. Fully grounding a symbolic state s_i means the state is mapped to a single world state w_i.

In a task plan, from a state s_i that is grounded to w_i, grounding a symbolic action a_i describes how the robot reaches the state s_{i+1} that follows.

To compute a trajectory for the transition $\langle s_i, a_i, s_{i+1} \rangle$ using a typical motion planner, the grounding process first constructs a motion planning query. The start configuration of the query has been determined by the configuration x that is part of w_i. Specifically, x is the configuration where the robot starts the task (i.e., $i = 0$), or is the end of the preceding trajectory, $\pi_{i-1}(1), i > 0$. The goal configuration can be computed from the definition of action a_i, the configuration space constraints (e.g., opening a drawer can require a specific range of orientation of the gripper), and the set of world states $\mathbf{G}(s_{i+1})$ that the system must to transition into, constrained by the symbolic transition itself. When such a query can be constructed, a sampling-based motion planner can generate π_i. As the robot moves along π_i, the corresponding world state also evolves. In the presence of inaccurate or incomplete modeling of the world used during planning, the motion planning query might not be possible to construct. When computed and executed in the real world, such a motion plan π_i can be unsafe or may achieve an end world state that is different from the planned w_{i+1}. If precomputed plans continue execution undeterred, differences between what was planned and what is executed will propagate along $\pi_j, j > i$.

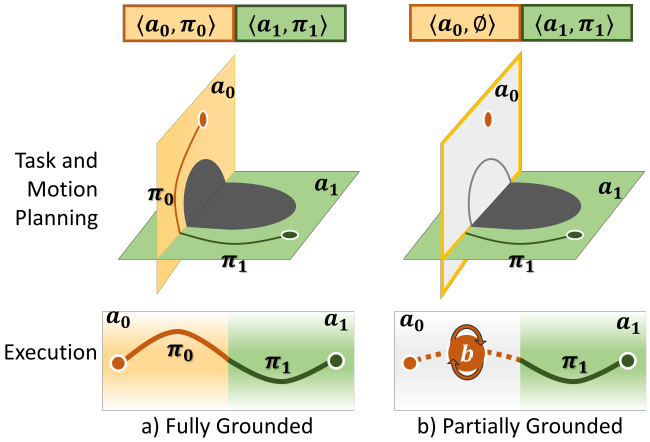

Fig. 4. The figure is a visual representation of grounding task and motion plans during planning and execution. The top row shows the task and motion plans. The middle visualizes the corresponding trajectories π_i in each configuration space. The bottom shows the corresponding trajectories in the real-world during execution. The colored spaces (yellow and green) represent individual symbolic states. **a)** In the fully grounded case, the real world closely matches the planning model, which is known. The is typical to classical TAMP methods. **b)** In the partially grounded case, the light gray regions indicate that we do not have good enough knowledge to ground action a_0. With a typical TAMP formulation, we cannot even plan the motion for a_0 due to the gap. With our work, we propose to leave a gap in the task and motion plan ($\langle a_0, \emptyset \rangle$, which is filled by a behavior **b** during execution (represented by the orange circle and dotted line).

For instance, the typical process of grounding a `Pick` action involves computing a grasp pose of the target object, then computing the goal configuration that achieves the grasp pose, and finally performing motion planning to this goal. When this process is part of a task and motion plan, the end world state after the trajectory π_i should satisfy $w_{i+1} \in \mathbf{G}(s_{i+1})$. If the exact pose of the object is not known at the beginning of the query, s_i, or it is hard to predict where the object ends up at the end of the query, s_{i+1}, this is liable to cause grounding failure. Another example involves pushing an object on an unknown surface. This may result in a stochastic end pose of the object that is different from what was planned.

Partial Grounding: A symbolic transition $\langle s_i, a_i, s_{i+1} \rangle$ in a task plan is partially grounded if either, **a)** a motion planning query cannot be constructed and computed, or **b)** the end world state of executing π_i can differ substantially from the w_{i+1} that was planned. Such a grounding failure corresponds to a grounding *gap* in the task and motion domain causing an empty trajectory ($\pi_i = \emptyset$) in a task and motion plan.

Behavior: A behavior **b** is a closed-loop module that controls the robot joints and perception systems and is capable of reasoning during execution. Each behavior corresponds to some symbolic action and if successful can execute real-robot

TABLE I
GLOSSARY OF TERMS

Terms	Definition
D_{TP}	Task domain
$\mathcal{S}$, s	Set of states and a state in task domain
$\mathcal{A}$, a	Set of actions and an action in task domain
Φ, ϕ	Set of constraints and a constraint in task domain
D_{MP}	Motion domain
$\mathcal{X}$, x	Configuration space and a configuration
π	Motion plan in configuration space
$\mathbf{T}$	Task & motion plan
$\mathcal{W}$, w	World state space and a world state
$\mathbf{G}$	Mapping from a symbolic state to set of world states
$\mathcal{B}, \mathbf{b}$	A set of behaviors and a behavior
$\overline{\mathbf{T}}$	Partially grounded task & motion plan

motions for a symbolic transition $\langle s_i, a_i, s_{i+1} \rangle$. A successful behavior must satisfy: $w_i \in \mathbf{G}(s_i)$ and the end world state in execution should satisfy $w_{i+1} \in \mathbf{G}(s_{i+1})$, i.e., the grounding gap is bridged, and the symbolic pre-condition and end-effect of a_i are satisfied. A set of behaviors is $\mathcal{B}$.

The behavior can be thought of as a local policy designed for a corresponding action. The general framework uses whatever behaviors are available and can be attempted during execution. The implementation can take the form of handcrafted subroutines, local policy modules, or general architectures like BTs [47]. They can also be learned controllers [48]. Some design choices are shown in Sec V. Note that the design of such behaviors is usually domain-specific, and is not the focus of this paper. The framework is formulated and presented to admit any of these design choices as long as the behavior is capable of using online information to compute and execute a motion for the action. We focus on how we leverage such behaviors with powerful TAMP solvers to overcome the grounding gaps, as elaborated in Sec. IV. An example on how a behavior might fill in a gap is shown in Fig. 4b.

Partially Grounded Task and Motion Plan: Given a task plan $(a_0, ..., a_n)$, where each a_i transitions from symbolic state s_i to s_{i+1}, and the task plan transitions from the initial state s_0 to a goal state $s_n \in \mathcal{S}_{goal}$, a partially grounded task and motion plan is

$$\overline{\mathbf{T}} = (\langle a_0, \pi_0 \rangle, ..., \langle a_n, \pi_n \rangle)$$

where actions that can be not grounded are allowed (i.e., $\pi_i = \emptyset$ is allowed). In order to be executed, a partially grounded plan requires

$$\forall \pi_i = \emptyset, \exists \mathbf{b} \in \mathcal{B} \text{ for } a_i$$

where a behavior is available for each action that is not grounded.

Task and Motion Planning for Execution in the Real (TAMPER): Given a task planning domain D_{TP}, a motion planning domain D_{MP}, a set of given behaviors $\mathcal{B}$, and a set of goal states $\mathcal{S}_{goal}$, TAMPER is the problem of interleaved computation and execution of partially grounded task and motion plans ($\overline{\mathbf{T}}$) consisting of a sequence of valid actions, motion, and closed-loop online behaviors that enable the robot to reach a goal state $s \in \mathcal{S}_{goal}$ in the real world.

IV. METHOD

We propose a framework for TAMPER capable of reasoning over partial groundings and real-world execution feedback to solve long-horizon tasks. The framework consists of a planning layer and an execution layer (Fig. 5). In planning, we use an existing TAMP planner to plan as much as we can using the current, incomplete model of the world to discover motion constraints and find a partially grounded plan with gaps. During execution, we leverage online behaviors to fill in gaps in the partial plan and feed back failures from execution as new constraints, completing only upon real-world success.

A. Foundations: Task and Motion Planning

Consider a standard incremental constraint-based task and motion planner in Alg. 1, introduced in previous work [1] It assumes a perfect model of the world is given. The input to the algorithm is the task planning domain, the initial state, the goal described as some symbolic expression, and the model of the world for motion planning.

Algorithm 1: TAMP [1]

```
   Input: D_TP, D_MP, s_init, S_goal, world
1  s ← s_init; Φ ← NoConstraint; T ← NoPlan;
   // Planning Loop
2  while T = NoPlan do
       // Task Planning
3      if T ← TP(s, S_goal, Φ, D_TP) fails then
4          if Φ = NoConstraint then
5              return NoSolution;
6          Φ ← NoConstraint;
       // Motion Planning
7      foreach a_i ∈ T do
8          if π_i ← MP(a_i, D_MP) succeeds then
9              T ← T ⊕ ⟨a_i, π_i⟩;
10         else
11             Φ ← Φ ∪ CONSTRAINT(a_i);
12             T ← NoPlan;
13             break;
   // Open-loop Execution
14 foreach ⟨a_i, π_i⟩ ∈ T do
15     EXECUTE(π_i)
16 return Success;
```

An SMT-solver as the task planner supports the incremental-solving feature, leveraged by many existing planners [1], [49], [50]. A candidate task plan is a sequence of symbolic actions (Alg. 1 line 3) which needs to be grounded. The grounding of action a_i creates a motion planning query from the current world state w_i, and computes a trajectory π_i using a sampling-based motion planner (Alg. 1 line 8). Motion planning can fail for some actions. These are then added as a symbolic motion constraint (Alg. 1 line 11) to the constraint stack of the SMT-based task planner, and a new task plan queried. The incremental solving feature maintains the guarantee of finding

a feasible plan with the minimum number of actions. If the task planner cannot find any alternate plans, it reports that the problem is either infeasible, or the constraints should be cleared to start over again with increased motion planning time budget (Alg. 1 line 12). When the planner successfully finds a task and motion plan $\mathbf{T}$, it is typically sent to the robot for *open-loop* execution (Alg. 1 line 15). This planner serves as a module in our proposed method, including necessary changes enabling it to work for grounding steps with knowledge gaps. The details are reflected in Alg. 2 and explained in Sec. IV-B.

B. TAMPER Framework

We now present our new algorithm (Alg. 2) for task and motion planning and execution reasoning (TAMPER).

We make the assumption that a set of behaviors corresponding to a subset of actions in the task domain is assumed to be available as input. Hence, the input to Alg. 2 is the task planning domain, the initial symbolic state s_{init}, the set of goal states that can be encoded as a symbolic expression, the model of the world, and a set of pre-defined behaviors. Our framework (Fig. 5) solves such a problem by interleaving planning and execution described as follows.

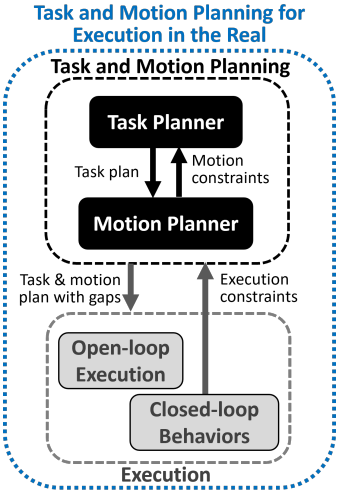

Fig. 5. The TAMPER framework.

1) Planning: The task planner computes a candidate task plan from s_{init} (Alg. 2 line 4). For each action in the plan, we first check if we can ground it using CANGROUND (Alg. 2 line 10), which is a typical TAMP process that creates motion planning queries. In manipulation problems, it usually involves first computing grasp poses of the objects, or placing poses of the objects, and then computing the robot configurations achieving these poses as the motion planning goal. Some examples of the grounding process of the actions that we use in the experiments are introduced in details in Sec. V. When CANGROUND outputs false, it is considered that this action cannot be grounded. For example, if the action is to pick up some object from the table, but the object is occluded from the sensor and its pose is unknown, CANGROUND should output false. Note that here we allow planning with incomplete models, so we consider that CANGROUND encodes the confidence on the model of the world. For instance, if the friction of a surface is inaccurately modeled, the consequence of a *push* might not be reliably planned.

If CANGROUND decides that the action can be grounded, the motion planning query is sent to a sampling-based motion planner (Alg. 2 line 11). If the motion planner fails to compute a trajectory, similar to TAMP, the failure is encoded as a symbolic constraint (Alg. 2, line 16), and pushed to the constraint stack to compute a new alternate task plan that does not include the particular action.

Algorithm 2: TAMPER

Input: $D_{\text{TP}}, D_{\text{MP}}, s_{init}, \mathcal{S}_{goal}, \mathcal{B}, world$

1 $s \leftarrow s_{init}$; $\Phi \leftarrow$ `NoConstraint`; $\overline{\mathbf{T}} \leftarrow$ `NoPlan`;
// Planning and Execution Loop
2 **while not** ISSATISFIED$(s, \mathcal{S}_{goal})$ **do**
// Partially-grounded TAMP
3 **while** $\overline{\mathbf{T}} =$ `NoPlan` **do**
// Task Planning
4 **if** $T \leftarrow \text{TP}(s, \mathcal{S}_{goal}, \Phi, D_{\text{TP}})$ ***fails*** **then**
5 **if** $\Phi =$ `NoConstraint` **then**
6 **return** `NoSolution`;
7 $\Phi \leftarrow$ `NoConstraint`;
// Grounding and Motion Planning
8 **foreach** $a_i \in T$ **do**
9 $\langle a, \pi \rangle \leftarrow \langle \emptyset, \emptyset \rangle$;
10 **if** CANGROUND$(a_i, world)$ **then**
11 **if** $\pi_i \leftarrow \text{MP}(a_i, D_{\text{MP}})$ ***succeeds*** **then**
12 $\langle a, \pi \rangle \leftarrow \langle a_i, \pi_i \rangle$;
13 **else if** HASBEHAVIOR$(a_i, \mathcal{B})$ **then**
// Behaviors for Failed Grounding
14 $\langle a, \pi \rangle \leftarrow \langle a_i, \emptyset \rangle$;
15 **if** $\langle a, \pi \rangle = \langle \emptyset, \emptyset \rangle$ **then**
16 $\Phi \leftarrow \Phi \cup$ CONSTRAINT(a_i);
17 $\overline{\mathbf{T}} \leftarrow$ `NoPlan`;
18 **break**;
19 $\overline{\mathbf{T}} \leftarrow \overline{\mathbf{T}} \oplus \langle a, \pi \rangle$;
// Execution with Behaviors and Feedback
20 **foreach** $\langle a_i, \pi_i \rangle \in \overline{\mathbf{T}}$ **do**
21 **if** $\pi_i \neq \emptyset$ **then**
22 EXECUTE(π_i)
23 **else if** $\mathbf{b} \leftarrow$ BEHAVIOR$(a_i, \mathcal{B})$ ***fails*** **or** $\overline{\mathbf{T}} \leftarrow$ REPAIR$(\overline{\mathbf{T}})$ ***fails*** **then**
24 $\Phi \leftarrow \Phi \cup$ CONSTRAINT(a_i);
25 **break**;
26 $s \leftarrow$ GETCURRENTSTATE$(world)$; $\overline{\mathbf{T}} \leftarrow$ `NoPlan`;
27 **return** `Success`;

If CANGROUND determines an action a_i cannot be grounded with the current model of the world, we check if any one of the given behaviors is applicable for a_i (Alg. 2 line 13). If a behavior is available, we add an empty trajectory to the task and motion plan (Alg. 2 line 14), indicating that the grounding of this action is deferred to the execution level. To continue grounding the next action a_{i+1}, we use an optimistic robot configuration to construct a safe world state heuristically in the planner as the start state of the next action. In our implementation, we design a repair strategy to rejoin to any of the configurations along π_{i+1} during execution (see the Sec. V-B for details). If there is no associated behavior, it is considered a grounding failure, and also encoded as a symbolic constraint (Alg. 2 line 16), to find a new task plan. The output of the planning part is a partially-grounded task and motion plan $\overline{\mathbf{T}}$, which allows gaps (empty trajectories $\pi_i = \emptyset$) to be

grounded by the behaviors in execution. Note that if the input $\mathcal{B} = \emptyset$, i.e., no behaviors are given, our planner reduces to a typical TAMP planner [1].

2) Execution: After a partially-grounded task and motion plan is computed, the execution layer executes the trajectories in sequence (Alg. 2 line 22). If π_i is empty, we execute the associated behavior for this action a_i. A behavior module may involve online sensing, planning, and execution. We assume a set of behaviors $\mathcal{B}$ are available as input. These can be simple sequential steps, BTs [47] designed by human experts, local policies that can be learned controllers [48], etc. The details of the behaviors that we use are discussed in Sec. V-B.

When the behavior succeeds, online trajectory repair is performed to join to the next pre-computed trajectory π_{i+1} using REPAIR (Alg. 2 line 23). Different repairing strategies exist for various replanning frameworks, and any strategy can be used here, as long as it joins to one of the configurations $\pi_{i+1}(\tau), \tau \in [0, 1]$. REPAIR returns the updated π_{i+1} to be executed next. If π_{i+1} was also empty, it will be handled by the next behavior, and REPAIR does not need to update it.

In this work, we do *not* assume that behavior execution and the REPAIR module always succeed. The behavior execution for a_i might fail because the necessary information to allow its successful execution may still not have been discovered after executing pre-computed trajectories or behaviors of all the actions from a_0 to a_{i-1}. This suggests that we might not be able to ground a_i from the current state, requiring to recompute a new task and motion plan. The REPAIR following executing the behavior of a_i may fail due to not finding a motion plan to the initial state of π_{i+1} (or other waypoints in π_{i+1}, defined by the actual implementation of REPAIR detailed in Sec. V-B) within the time limit. This indicates that the execution of the behavior of a_i succeeded in the real world, but the resulting state of the world might make the next action a_{i+1} not feasible. When any of such execution failures happen, similar in spirit to incremental TAMP, we can encode them as symbolic constraints (Alg. 2 line 24) for replanning — as opposed to typical TAMP which replans within the planning loop, we are *replanning within a planning and execution loop*. We infer the current state by combining the previous knowledge as well as new information from sensing (Alg. 2 line 26). Then, we return to the planner layer, and replan from the current state (Alg. 2 line 4). Replanning uses all the constraints we have discovered from the planning and execution so far, improving the efficiency of solution discovery and execution.

It is worth noting that in the algorithm we use a symbolic state s to track the world state, only to keep it concise. All the state variables (e.g., object poses) are essentially kept track of throughout the framework for both planning and execution.

Note on Planning vs. Execution Trade-off: For a typical TAMP planner [1], it always calls the task planner to find an alternate candidate plan if it fails to ground an action. In our proposed algorithm, we prioritize using given behaviors when an action cannot be grounded, rather than relying on the task planner to compute another candidate plan (Alg. 2 line 13). Similar to TAMP methods [1], [2], [11], a model of the world (domain) is available to plan over within TAMP as well as in the behaviors. If the execution is inconsistent with the domain and errors in the pipeline induce unrecoverable dead-end states (where the goal becomes unreachable), such execution will trigger task failures. Note the proposed method still operates within an available and accurate task domain, while expanding the class of problems we can address using an enhanced paradigm built on classical TAMP. This trade-off is discussed in details in Sec. VII. A discussion on theoretical guarantees such as probabilistic completeness in the domain combining planning and execution is also presented in Sec. VII.

C. Necessary Assumptions and Modules

We present the considerations that our proposed framework relies on to extend the applicability of traditional TAMP (Alg. 1), where TAMP typically assumes open-loop execution [1], [2], [51]. Notably, many of these assumptions are not unique to the proposed approach and are known to come up when typical TAMP methods such as Alg. 1 have been implemented in practice [6], [52]. We argue that these assumptions are reasonable and necessary because without our framework, the class of problems we study in this paper either cannot be addressed at all, lead to infeasibility, or yield poor empirical performance (as demonstrated in Sec. VI).

- We assume the set of behaviors is provided as input accompanying the specific problem domain. How the behaviors can be more efficiently designed or learnt based on the execution results of the proposed framework is left for future work with some discussion in Sec. VII.
- Each symbolic action is associated with provided domain knowledge on an evaluation of the CANGROUND function. This knowledge is encoded as a part of CANGROUND. The availability (and success) of such a module is a common assumption in typical TAMP methods, which rely on robust or well-modeled perception and execution modules. It is either the case that the world state is known and actions are assumed to be deterministic [1], [51], [2], or it is assumed the robot knows that stochasticity exists in the domain and the TAMP problems are solved based on some known models of uncertainty [11], [13], [4]. In this work, we can address problems where the perception fails or when models of uncertainty are unknown or inaccurate. This is possible using execution-level behaviors that are available and performant across these grounding failures. The flexibility of the proposed framework comes at the behest of having access to the CANGROUND function that *knows what is not known* or cannot be grounded. Typically such information might already be readily available in TAMP implementations — consider the case where a pose estimator is invoked for an object in the planning domain and reports no detection, or the case where no robust simulator exists for grounding a symbolic action. Such failures occur but are not addressable by a standard TAMP framework.
- The framework needs a REPAIR module during execution to be deployed after the completion of a behavior to rejoin the precomputed task and motion plan. Our experiments show that this module typically managed to use motion planning to *rejoin* the subsequent trajectory. When it fails, the framework reverts to TAMP replanning [5] within the high-level loop of Alg. 2.

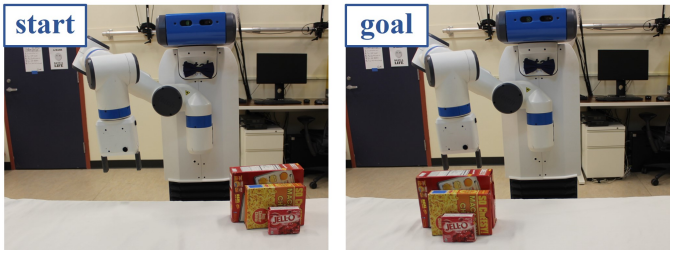

Fig. 6. **Horizontal Stacking** Benchmark: The task is to move the objects to the goal positions. The initial poses of the objects are designed to have some occlusion. Strict ordering constraints exist because smaller objects cannot be grasped in the presence of a nearby larger object.

Fig. 7. **Kitchen Arrangement** Benchmark: Move the objects to the goal positions. The starting position of one of the objects is inside the drawer. In this case, there is both occlusion and uncertainty in execution outcomes.

V. Real-World Benchmarks and Design Choices

We focus on the evaluation of our method on real-world scenarios. In this section we introduce two scenarios and our necessary design choices dictated by the target scenarios.

A. Benchmarks

Our real-world experiments are performed on a Fetch robot with an RGBD sensor mounted on its head and objects from the YCB dataset [53] and the HOPE dataset [54]. Vicon Cameras are used for pose estimation of the obstacles such as the table and the drawer. A deep learning-based perception module (DOPE [55]) is used to detect the objects. The task planner is implemented using Z3 planner [7], and the motion planner is RRT-Connect [56] via Robowflex [57].

1) Horizontal Stacking: As shown in Fig. 6, there are three objects on the table with known geometric models. The initial object poses are designed to create occlusions for the camera. The goal is to move all the objects to the goal positions, which are explicitly specified to allow only one order of placement. If a large object is placed to its goal position first, the geometry blocks the robot arm from placing a smaller object to its goal position. The available actions are `Pick` and `Place`.

2) Kitchen Arrangement: We add a drawer and a shelf to the scene (as in Fig. 7). Initially, one object is in a closed drawer, and two objects are on the table. When the small object is on the shelf, it can only be picked from its side. When the large object is close to the shelf, it cannot be picked from any direction. To address this challenge we define a `Push-Pick` action. Grounding it involves first computing the motion to push an object away with a distance, and then computing the motion of picking. We also define an `Open` action that opens a drawer, where grounding requires computing trajectories to approach and pull the drawer handle. The goal is similar to the Horizontal Stacking, but on the shelf. The available actions are `Pick`, `Place`, `Open` and `Push-Pick`.

B. Framework Design

Here, we particularly focus on how we design the execution layer of the framework. Note that the following design choices are modular and can be replaced by other options. Such design efforts help to address realistic problems.

1) Symbolic Encoding: Some previous works assume a discretization over the workspace into grid locations [1], [49], where each location is associated with a different symbol, which makes the state space unnecessarily large. We choose to only explicitly encode the regions/locations of interest into different symbols, which is a common choice in TAMP literature [51], [11], [2]. For a large support surface like the table, we do not discretize it but instead only assign a single symbol to it. Such an encoding is also helpful to gain efficiency dealing with partial models. For example, even the pose of an object is unknown, its symbolic state can be captured by a single predicate OnTable(). We encode the essential manipulations such as `Pick`, `Place`, `Push`, and `OpenDrawer` as symbolic actions. Sensing is not explicitly encoded as an action, even though it is critical for the benchmark problems that have partial groundings due to occlusions. Instead, sensing is performed online in the behaviors, which avoids inflating the state and action space. Our framework balances the trade-off of designing a more complicated abstraction model versus designing a more powerful local behavior. Which one is better depends on the actual problem to be solved.

2) Occlusion Model: Since we purely rely on point cloud sensors to get geometric information of the real world, we address having access to partial observations due to occlusions, etc. To perform motion planning in such real-world manipulation problems, naive strategies can either assume there is nothing to avoid in the occluded area (most optimistic), or to perform collision-checking assuming the occluded area is fully occupied by something (most conservative).

In our experiments, we use a parametric model of the occlusion inspired by [58]. Given an input point cloud $\mathbb{C}$, we perform ray-casting of the points from the camera viewpoint to get a new cloud $\mathbb{C}_r$ where the occluded parts are also filled with points. Then, we crop out the points that correspond to the known objects (the objects that are recognized by the perception modules), and get the clusters that correspond to each object $o \in \mathcal{O}$. Then, the occluded geometry caused by an object o_1 can be represented by the set of clusters $\{\mathbb{C}^1_{o_1}, ..., \mathbb{C}^m_{o_1}\}$. We compute the centroid of each cluster, and shrink the cluster with ratio $\epsilon_{o_1} \in [0, 1]$. When the ratio $\epsilon_{o_1} = 0$, it is the most optimistic representation, as it means there is nothing in the occluded area. When $\epsilon_{o_1} = 1$, it is the most conservative representation, assuming the area is fully occupied. We use the same $\epsilon = 0.6$ for all occlusions. Fig. 8 (top) shows the side view of our occlusion model with three different ϵ. Fig. 8 (bottom) shows a visualization of our occlusion model in occupancy grids [59]. Such an occlusion model is used in both the planning and the execution layer.

3) Grounding Modules: There are often cases that a task plan requires placing objects at intermediate locations. Choos-

ing the placement poses effectively and feasibly is important in highly symbolically constrained problems. We encode the whole table surface as a region with a single symbol associated. Therefore, we need a placement location sampler to ground the `Place` action. We perform uniform sampling within the robot's reachability area with certain constraints that take into account our occlusion model—avoiding intersections between the placement pose's occlusion model and the point cloud's occlusion model. To get configuration space goals, we sample valid inverse kinematic (IK) solutions, where we use the standard IK solver in MoveIt [60]. The grasping problem is not the focus of the work. Where the object mesh is available, predefined axis-aligned grasps of objects are used. When any of grounding modules cannot find a feasible value with a given time limit, it is considered as grounding failure and should be encoded as a symbolic constraint.

4) Behaviors: All of the following behaviors are expert-designed, but they can also be learned policies, etc. Each behavior is executed step by step. If the execution of any step fails, it stops execution, and the failure is encoded as a constraint to trigger task-level replanning (Alg. 2 line 24).

We use the following behavior for `Pick` in all benchmarks.

`Pick` Behavior

1. Invoke DOPE [55] to detect the pose of the object.
2. If the object that we want to pick is detected compute a grasp, otherwise return failure.
3. Perform motion planning online and proceed to execution to grasp object.

We use the following behavior for the `Push-Pick` action in the *Kitchen Arrangement* benchmark, which first pushes the object and then picks it up.

`Push-Pick` Behavior

1. Perform online constrained motion planning and execution to horizontally push the object along one of its axes with a fixed distance.
2. Use DOPE to detect the pose of the pushed object.
3. Compute axis-aligned grasp from its updated pose.
4. Perform motion planning online and proceed to execution to grasp object.

For the behavior of `OpenDrawer`, we design a sequence of key configurations parameterized by the pose of the drawer and plan and execute it online. The behaviors can be implemented in several ways. For the above behaviors, we use sequential scripts. We use Behavior Trees [47] to implement the behavior in the demo described in Sec. VI-C. If the behavior fails, a constraint is fed back to the framework to compute a new partially grounded task and motion plan (Alg. 2 line 4).

5) Repairing Strategy: As discussed in Section IV, after the execution of a behavior $\mathbf{b}(a_i)$ of action a_i, we try to join to the pre-computed trajectory π_{i+1} of the next action. Any repairing strategy [61] can be used as long as it plans for a path to one of the configurations in π_{i+1}, i.e., $\pi_{i+1}(\tau), \tau \in [0, 1]$. Note that

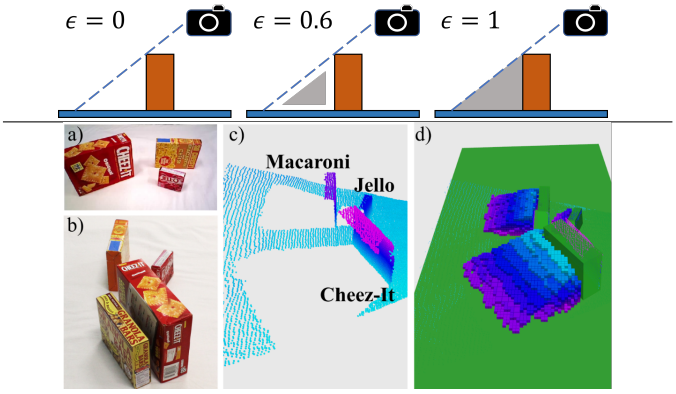

Fig. 8. (**Top**) Our parameterized occlusion model when it is (**top-left**) most optimistic, with $\epsilon = 0$, (**top-middle**) parameterized by $\epsilon \in [0, 1]$, and (**top-right**) the most conservative $\epsilon = 1$. (**Bottom**) shows the real RGBD sensor. **a)** RGB camera input. **b)** The side view of the scene (not available to robot). **c)** Point cloud detected by the camera (rotated to illustrate the occlusion). **d)** An example of our occlusion model shown in octomap with $\epsilon = 1$.

back in the task and motion planning stage, an initial state must be constructed to plan for π_{i+1}. Typically, any feasible state can be used since we only need to rejoin to one of the configurations in π_{i+1}, not necessarily only to $\pi_{i+1}(0)$. We choose to build the state from a robot configuration where the links are away from the domain-specified task space region, which is effective in practice. Fig. 14a shows an instance where the arm is raised away from the drawer and the table.

Our REPAIR strategy is to always try to join the first configuration $\pi_{i+1}(0)$. If this fails, then plan to rejoin $\pi_{i+1}(1)$. It is possible that the geometries of the world change sufficiently and repair fails (or subsequent trajectories become infeasible). Failure here triggers replanning with the updated state.

VI. RESULTS

In this section we report our empirical performance using data which has been exclusively collected from real-robot experiments. More than 40 real-world physical trials have been used to compare our method with a replanning baseline. It has to be stressed that the each experiment involves many-step planning and execution attempts, as well as multiple replanning tries upon sensed environment updates. Real-robot trials prove the most accurate, albeit more challenging and time-consuming, reflection of performance, where we demonstrate clear performance benefits of our approach.

To promote replicability in other real-world setups as well as simulated ones, we share the experiment setup information from our benchmark trials [1], including object poses, point clouds, RGB images, and domain models. This is included in an open-source dataset directed towards real-robot TAMP applications, which we believe is one of the first of its kind.

A. Evaluated Methods

*1) **(Base)** Closed-loop TAMP Algorithm:* This baseline is implemented based on the method in [5]. The overall goal is given, which may involve objects that are not observed yet. It extracts a subgoal that only involves the observed objects,

[1]Videos of all experiments are at: https://kavrakilab.org/2023-pan-tamper/

and greedily plans for this subgoal. Each time a TAMP plan is executed, it calls the perception module again, and plans for a new subgoal given new observations, until the task is completed. To ensure a fair comparison the baseline shares the same modules as the proposed method wherever it is used, like the task planner, motion planner, sampler, etc.

*2) **(Ours)** TAMPER:* The proposed framework and design choices described in Sec. IV and V.

B. Benchmark Results

Fig. 11 summarizes the start and end of the 10 problems in Horizontal Stacking, and Table II shows the planning time and real-world execution time. The first row (Plan-BE) shows how much time is spent in task and motion planning prior to the execution stage (i.e., the planning loop presented in Alg. 2 from line 3 to 19 encountered the first time). The second row shows how much time is devoted to planning after execution starts. It consists of both a) task and motion replanning, if triggered, and b) motion planning within the behaviors and the REPAIR module (Alg 2 line 23). The third row shows how much time is devoted to sensing modules after the execution begins. The fourth row exhibits the total time spent on executing trajectories (both the pre-computed ones and the online-computed ones in the behaviors) on the robot controllers. The fifth row is the sum of the numbers shown in the four rows above. The last two rows show the number of object interactions during solving the problem instance and whether it successfully accomplished the goal. In this problem, the placement of the objects introduces ordering constraints since the robot cannot interact with the smaller objects while the larger objects are nearby. The relative size and position of the objects also introduce gaps due to occlusion. In the 10 scenarios, the proposed method executed 9.2 ($\pm$ 0.98 one standard deviation) symbolic actions on average with 1.1 ($\pm$ 0.3) behaviors to complete the task. For all the problem scenarios, the proposed method takes much less computation during execution, and has fewer object interactions, with the overhead of slightly higher planning time before the execution starts. Counting sensing, planning and execution, the total time to finish each task is much less compared to the baseline.

This demonstrates our advantage of leveraging TAMP with even an incomplete model of the real world, which discovers motion constraints during planning instead of executing expensive real-world motions to find the same set of constraints.

The start and end states reached by the proposed method over 10 problems in Kitchen Arrangement are shown in Fig. 12 and the performance is recorded in Table III. Here, one of the object poses is unknown because it is inside the drawer, creating a gap. Another object is close to the edge of the shelf making it not possible to pick it directly. A `Push-Pick` action is also made available, that first pushes the object away from the shelf and then attempts to pick it up. Note that the baseline encounters catastrophic failure for all the scenarios. The reason is that the end state of pushing the object cannot be simulated perfectly in the planning stage. Any gaps here introduce undesirable interactions along subsequent actions. Since the baseline only executes the computed motion plans sequentially, when the pose of the object is different than what was computed, the trajectory may be in collision (see Fig. 9).

The proposed method leverages an online `Push-Pick` behavior to achieve robustness of the execution of this action. The proposed method succeeds in all the problems. In the 10 scenarios, the proposed method executed on average 8.4 ($\pm$ 0.91) symbolic actions with 2.3 ($\pm$ 0.46) behaviors.

Note: The initial poses of the objects (Fig 11 and 12) are manually designed to create challenging task-motion interactions and gaps. To ensure fair comparisons the poses are recorded and if the same setup is evaluated again, the poses are manually lined up by overlaying the point cloud and the recorded poses. This process is similar to the replication steps alluded to in the dataset discussion.

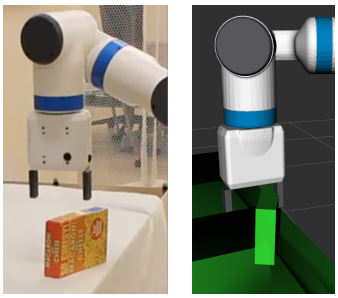

Fig. 9. An example of failure the baseline encounters with the Kitchen Arrangement benchmark. For safety, this trajectory was not executed. **Left**: The moment before the failure, side view. **Right**: The catastrophic failure of gripper-object collision that would have happened, replicated in RViz from the plan and object pose detection.

Example real-robot runthroughs of our proposed method on the Horizontal Stacking and the Kitchen Arrangement benchmarks are shown in Fig. 13 and 14.

C. Grocery Demo

Fig. 10. The robot doing groceries. The objects must be placed into the large bag on the right, with the eggs stacked on the apples. Which paper bag contains which object cannot be identified from the initial state. The robot has to search for and scan the barcode on one surface of the bag.

In this demo, we show that our framework can deal with yet another kind of partial grounding in task and motion planning problems. Initially, we have two paper bags containing different kind of groceries (see Fig. 10). The goal is to move them into the large paper bag. To avoid crushing the eggs, the bag of apples cannot be dropped on top of the eggs. Since the two bags look the same, the robot cannot distinguish between them. We devise a barcode which can be scanned to deduce whether it is eggs or apples. We used the following behavior, implemented as a BT, to enable the robot to check and grasp a bag.

`Pick-Point-Cloud` Behavior

1. Segment out the point cloud of one paper bag.
2. Compute grasp from point cloud using GPD [62].
3. Move the bag closer to the camera, and keep rotating it until the barcode (AprilTag [63]) is detected.
4. If the barcode matches the target object, the grasp action succeeds, otherwise put this bag back and repeat the above for the next point cloud cluster.

This behavior serves as a local policy that fully grounds `Pick` during execution. The proposed framework successfully completes the task by using the behavior.

TABLE II
HORIZONTAL STACKING BENCHMARK (BE IS BEFORE EXECUTION COMMENCES, AE IS AFTER EXECUTION COMMENCES)

	Problem 1		Problem 2		Problem 3		Problem 4		Problem 5		Problem 6		Problem 7		Problem 8		Problem 9		Problem 10		Mean	
	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base
Plan-BE (s)	0.83	0.14	0.69	0.14	7.50	3.24	9.42	0.24	0.76	0.12	0.74	0.09	6.91	3.27	7.28	3.21	7.42	0.23	0.74	0.12	4.23	1.08
Plan-AE (s)	9.22	21.83	9.15	17.62	5.01	18.61	0.18	2.30	9.07	17.18	8.93	23.15	5.06	18.08	0.12	19.59	0.16	2.34	6.91	7.86	5.38	14.86
Sense-AE (s)	5.38	7.06	6.41	3.54	2.36	3.42	0.82	3.75	6.41	6.68	6.17	3.64	2.29	2.71	0.77	3.18	1.02	3.04	4.88	4.20	3.65	4.12
Exec (s)	94.23	172.68	95.19	110.94	94.37	144.19	75.19	99.34	106.65	166.67	95.39	112.74	98.32	151.05	82.46	134.49	76.77	92.87	81.37	94.65	89.99	127.96
Total (s)	109.66	201.71	111.44	132.24	109.24	169.46	85.61	105.63	122.89	190.65	111.23	139.62	112.58	175.11	90.63	160.47	85.37	98.48	93.9	106.83	103.25	148.02
#Grasps	5.00	9.00	5.00	6.00	5.00	8.00	4.00	5.00	5.00	9.00	5.00	6.00	5.00	8.00	4.00	8.00	4.00	5.00	4.00	5.00	4.6	6.9
Success	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig. 11. The 10 problems in Horizontal Stacking. Each column shows the start (top) and the end (bottom) of TAMPER's execution in each problem.

TABLE III
KITCHEN ARRANGEMENT BENCHMARK (BE IS BEFORE EXECUTION COMMENCES, AE IS AFTER EXECUTION COMMENCES)

	Problem 1		Problem 2		Problem 3		Problem 4		Problem 5		Problem 6		Problem 7		Problem 8		Problem 9		Problem 10		Mean	
	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base
Plan-BE (s)	7.97	11.60	6.37	10.37	8.08	12.80	9.59	11.67	11.11	9.00	41.37	14.63	35.91	12.49	34.18	23.01	12.56	7.61	12.52	8.08	17.97	12.13
Plan-AE (s)	26.71	0.00	23.46	0.08	32.41	0.08	0.30	0.00	0.27	0.00	1.40	0.08	0.35	0.10	0.32	0.08	0.29	0.00	1.63	0.00	8.71	0.04
Sense-AE (s)	6.56	0.00	5.28	0.00	5.64	0.00	2.66	0.00	1.98	0.00	2.55	0.00	2.61	0.00	2.37	0.00	2.83	0.00	2.41	0.00	3.49	0.00
Exec (s)	91.09	23.68	96.61	30.92	95.92	29.46	102.41	13.63	106.99	22.30	96.51	59.05	95.48	57.81	95.19	69.87	114.84	13.58	108.40	13.79	100.34	33.41
Total (s)	132.33	35.28	131.72	41.37	142.05	42.34	114.96	25.3	120.35	31.3	141.83	73.76	134.35	70.4	132.06	92.96	130.52	21.19	124.96	21.87	130.51	45.58
#Grasps	5.00	1.00	5.00	1.00	5.00	1.00	5.00	1.00	5.00	1.00	4.00	2.00	4.00	2.00	4.00	2.00	5.00	1.00	5.00	1.00	4.7	1.3
Success	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗

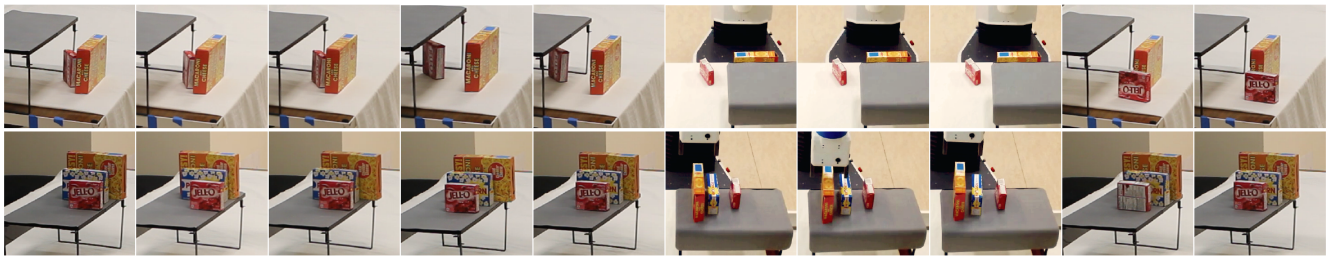

Fig. 12. The 10 problems in Kitchen Arrangement. Each column shows the start (top) and the end (bottom) of the execution of TAMPER in each problem. Since the execution outcome of the push is stochastic, we run each scenario multiple times. From left to right, columns 1-3, 4-5, 6-8, and 9-10 show four different starting poses with either 3 runs or 2 runs each. Note that in all scenarios, one object is initially inside the drawer.

D. Real-Robot TAMP *and Execution Dataset*

We provide our real world benchmarking pipeline as well as a dataset of our runs of all the benchmarking problems [2]. The Robot Operating System (ROS) [64] is required to run our pipeline. We built a pipeline to record the poses of all the involved objects and obstacles that are relative to the robot frame. To replicate a problem with the real robot, we can visualize the initial poses of the objects in RViz [65] in the robot frame. For any object without an accurate model, a bounding box is necessary to replicate its pose. Finally, we need to manually move around the real-world objects and/or the robot to match the published objects. We recorded the real time running data of our method on the benchmarking problems discussed in this section. The data are ROS-bags that can be replayed in simulation.

[2]Available at: https://github.com/KavrakiLab/tamper-data

VII. DISCUSSION

This work introduces a task and motion planning for execution in the real (TAMPER) framework that addresses realistic scenarios where the symbolic domain might be known but the geometric knowledge of the world has gaps which often cannot be resolved before execution begins. This work has pushed the notion of the TAMP paradigm into the execution layer by addressing the incomplete TAMP domain during planning. The power of traditional task planning and motion planning is applied wherever information is known, and a partially grounded task and motion plan is constructed with gaps. These gaps are bridged during execution using closed-loop behaviors, which are assumed to be input and can be implemented using hand-crafted subroutines, behavior trees, or learned controllers. Information can then be fed back as execution constraints to the framework to potentially replan.

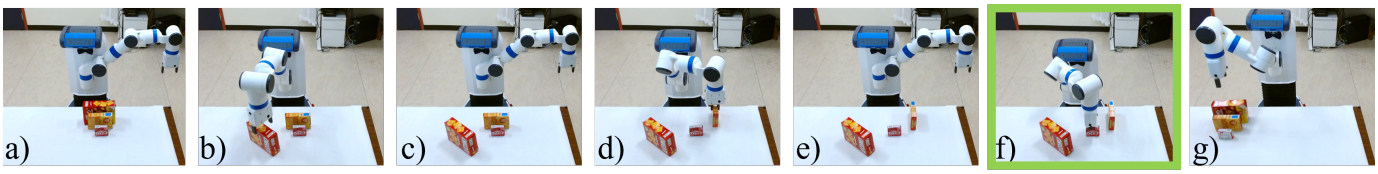

Fig. 13. An example run of the proposed method on the Horizontal Stacking benchmark. A successful use of an online behavior is highlighted in green. **a)** The robot plans with only sensing the front object, discovers the geometric constraints of the goal (i.e., planning with the object models, the task and motion planner finds that a smaller object cannot be placed to its goal after a bigger object is already placed there), and finds out that the behavior of picking up the other two objects fails from this state because they cannot be observed by the head camera. **b)** The robot replans with the constraints and executes the first action of moving the front object. **c)** The behavior of picking up the small object fails again in execution because it is still occluded by the middle object and cannot be observed by the head camera. **d)** The robot replans again, and moves the middle object away. **e)** The start state of the behavior for picking up the small object. Here the small object can be observed by the head camera. **f)** The online execution of the behavior for picking up the small object succeeds and the behavior finishes with the robot grasping the object. **g)** The robot successfully rejoins to the pre-computed trajectory of the next `Place` action. Then it proceeds to the open-loop execution of pre-computed trajectories to move the other objects to the goal.

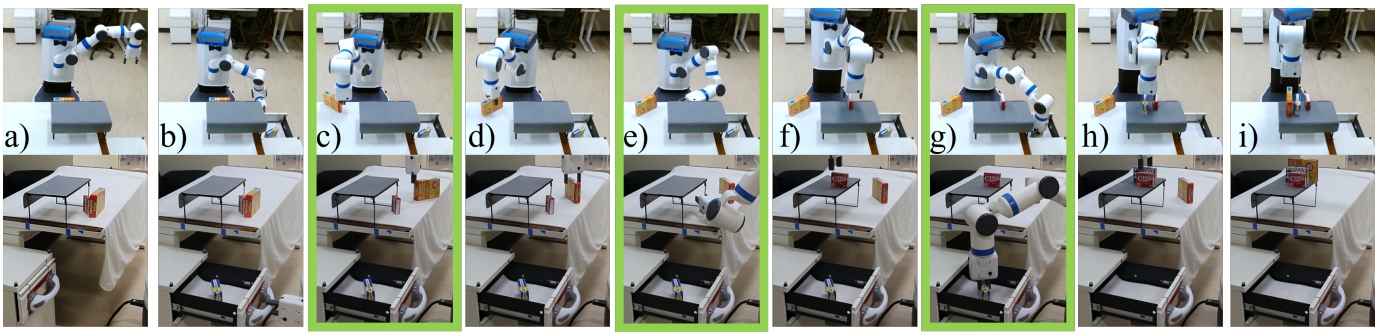

Fig. 14. An example run of the proposed method on the Kitchen Arrangement benchmark. The successful uses of an online behavior are highlighted in green. **a)** The robot plans with only sensing the front object, discovers the geometric constraints of the goal, and finds out that the behavior of picking up the small object fails from this state. **b)** The robot opens the drawer. **c)** The behavior of picking up the small object fails again in execution, as it is still not revealed. The robot then replans and executes the online behavior of pushing and picking up the front object. **d)** The robot executes the pre-computed trajectory of placing it at an intermediate location. **e)** Online execution of the behavior for picking up the small object which is now revealed. **f)** Open-loop execution of placing the small object on the shelf. **g)** Online execution of the behavior of picking up the object from the drawer. **h-i)** Open-loop execution of pre-computed trajectories to move the objects to the goal.

The TAMPER framework overcomes the limitations of traditional TAMP solvers, which need all the information about the TAMP domain to be known during planning, as well as more recent techniques which either decouple the task-motion problem, or only address fully defined but partial TAMP domains incrementally using replanning. The real-world data is accumulated during our experiments is also being shared in the form of a dataset to allow broader accessibility to realistic problems and operational sensing data, and enable researchers to recreate or evaluate the studied problems.

The current work also opens up a lot of interesting points of discussion. The proposed framework has been presented from the point of view of expanding the domain of applicability of traditional TAMP methods, which have grown popular as a powerful and general long-horizon planning paradigm. To endow the power of execution robustness, the availability of specialized behaviors is assumed as input. Although this has been demonstrated to be reasonable for a suite of benchmarks and demonstrations, it is definitely an open problem— how can behaviors be generated automatically; how can we actively discover the needs of generating the behaviors during real-world execution; how can behaviors be evaluated; what is the best way to model behaviors; how many behaviors are sufficient? The current work opens the door to addressing these questions in the future towards more complex compositional, extensible, and targeted robotic frameworks that are designed to be effective in the real world.

Our framework shows a trade-off between re-computing a new task plan upon grounding failure and leaving it as a gap in the task and motion plan to be filled by a behavior. We choose to prioritize using a behavior to fill the grounding gap rather than finding a new task plan. Note that although the execution of the behaviors is not guaranteed to succeed, such constraints can only be discovered after real-world executions, and discovering them may be crucial to eventually solving the problem. However, there can be cases where computing and grounding an alternate task plan is more efficient than always committing to the behavior. For example, if grasping any of the two objects fulfills the goal, where one is occluded and one is not, grounding the alternate task plan of grasping the visible object is much more efficient than trying to use a behavior to grasp the occluded object and failing. Whether to prioritize planning or execution effort depends on the specific problem setting. If the behaviors are expected to be reliable or that specific action is critical to the feasibility, it might be better to always try executing a behavior. If there are several feasible task plans, alternative solutions that do not involve behaviors can be searched. However, this can be expensive in itself and is liable to lead to suboptimal task plans. This trade-off is subject to the needs of the problem.

Another question is the modeling of uncertainty. Gaps can be seen from the lens of instances of belief uncertainty modeled in the world. It is of definite interest to explore how more informed modeling of the underlying uncertainty can

be better utilized while preserving performance. For example, even though it still remains an open problem how to efficiently model long-horizon manipulation tasks as POMDPs, it would be interesting to explore how can we combine the proposed framework which focuses on long-horizon tasks, with the general POMDP formulation to deal with uncertainty in the real world in an informed and structured way.

The present work serves as an augmentation of traditional algorithmic planning approaches into realistic domains, which have been recently dominated by learning-based techniques, due to their robustness to real sensing data. Through the behaviors used in the current work, hooks for incorporating powerful learning-based grounding mechanisms as well as local controllers are provided within a high-level framework. This opens up investigations into the role of learning-based methods when designed specifically for the proposed paradigm of robots that reason about tasks, motions, and execution.

An important aspect of traditional TAMP methods has been their theoretical soundness and guarantees. Typical asymptotic guarantees which practically demonstrate convergence properties to either feasibility or optimality become ill-posed when execution in real-time is introduced. As an example, typical TAMP methods can achieve probabilistic completeness, avoiding unrecoverable states by e.g., backtracking [1], [2], [11], [51]. Such states have to be modeled in the problem domain to allow reasoning over them. If execution failures happen leading to unmodeled dead-end states (e.g., glass cup fallen and broken to pieces), the typical TAMP methods cannot recover and have to report failure. In this sense, we hold similar assumptions that our framework avoids the modeled catastrophic failures by leveraging the provided behaviors (e.g., the Kitchen Arrangement benchmark in Fig. 9). If dead-end states exist that are not modeled by our TAMP module or by the behaviors, we can also only report failure, similarly to typical TAMP methods. On the other hand, if we assume that the execution of all behaviors do not lead to irreversible states in the real-world, the proposed framework achieves probabilistic completeness if all the planning modules used are probabilistic complete. For the planning and execution problem as a whole to express reasonable guarantees, there is a need to rethink the kind of finite-time properties we can leverage out of traditional techniques.

The wealth of exciting directions the present work opens up is testament to the promise held by pushing the applicability of robotic long-horizon reasoning to completing tasks within real world execution. The proposed TAMPER framework is a significant stepping stone towards achieving this goal.

REFERENCES

[1] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: integrating symbolic planners and Blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[3] M. Toussaint, "Logic-geometric programming: an optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[4] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5678–5684.

[5] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett, "Long-horizon manipulation of unknown objects via task and motion planning with estimated Affordances," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 1940–1946.

[6] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 265–293, 2021. [Online]. Available: https://doi.org/10.1146/annurev-control-091420-084139

[7] L. d. Moura and N. Bjørner, "Z3: an efficient SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[8] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[9] Z. Kingston, A. M. Wells, M. Moll, and L. E. Kavraki, "Informing multi-modal planning with synergistic discrete leads," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3199–3205.

[10] Z. Kingston and L. E. Kavraki, "Scaling multimodal planning: using experience and informing discrete search," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 128–146, feb 2023.

[11] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9285–9291.

[12] M. Wells, Z. Kingston, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Finite-horizon synthesis for probabilistic manipulation domains," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6336–6342.

[13] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "Failure is an option: task and motion planning with failing executions," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1947–1953.

[14] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[15] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: online POMDP planning with regularization," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013.

[16] H. Kurniawati and V. Yadav, "An online POMDP solver for uncertainty planning in dynamic environment," in *Robotics Research: the 16Th International Symposium ISRR*. Springer, 2016, pp. 611–629.

[17] J. K. Li, D. Hsu, and W. S. Lee, "Act to see and see to act: POMDP planning for objects search in clutter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5701–5707.

[18] Y. Xiao, S. Katt, A. ten Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8241–8247.

[19] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[20] A. Adu-Bredu, N. Devraj, P.-H. Lin, Z. Zeng, and O. C. Jenkins, "Probabilistic inference in planning for partially observable long horizon problems," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3154–3161.

[21] C. Phiquepal and M. Toussaint, "Combined task and motion planning under partial observability: an optimization-based approach," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9000–9006.

[22] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, may 2019, pp. 8993–8999, (Best paper award in Cognitive Robotics).

[23] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI'13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. Association for Computing Machinery, 2013, pp. 854–860.

[24] M. Colledanchise and P. Ogren, *Behavior trees in robotics and al: an introduction*, 1st ed. USA: CRC Press, Inc., 2018.

[25] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 618–12 624.

[26] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[27] G. D. Giacomo, R. Reiter, and M. Soutchanski, "Execution monitoring of high-level robot programs," in *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR'98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 453–465.

[28] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, H. Zender, G.-J. Kruijff, N. Hawes, and J. L. Wyatt, "Robot task planning and explanation in open and uncertain worlds," *Artificial Intelligence*, vol. 247, pp. 119–150, 2017, special Issue on AI and Robotics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437021500123X

[29] G. Coruhlu, E. Erdem, and V. Patoglu, "Explainable robotic plan execution monitoring under partial observability," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2495–2515, 2022.

[30] S. Lemai and F. Ingrand, "Interleaving temporal planning and execution in robotics domains," in *Proceedings of the 19th National Conference on Artificial Intelligence*, ser. AAAI'04. AAAI Press, 2004, p. 617–622.

[31] P. Ögren and C. I. Sprague, "Behavior trees in robot control systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 81–107, 2022.

[32] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8839–8845.

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

[34] C. Pezzato, C. H. Corbato, S. Bonhof, and M. Wisse, "Active inference and behavior trees for reactive action planning and execution in robotics," *IEEE Transactions on Robotics*, pp. 1–20, 2022.

[35] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: learning to generalize across hierarchical tasks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3795–3802.

[36] K. French, S. Wu, T. Pan, Z. Zhou, and O. C. Jenkins, "Learning behavior trees from demonstration," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7791–7797.

[37] N. Gopalan, E. Rosen, G. Konidaris, and S. Tellex, "Simultaneously learning transferable symbols and language Groundings from perceptual data for instruction following," *Robotics: Science and Systems XVI*, 2020.

[38] R. Patel, E. Pavlick, and S. Tellex, "Grounding language to non-Markovian tasks with no supervision of task specifications," in *Proceedings of Robotics: Science and Systems*, 2020.

[39] Y. Jiang, F. Yang, S. Zhang, and P. Stone, "Task-motion planning with reinforcement learning for adaptable mobile service robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7529–7534.

[40] M. Noseworthy, I. Brand, C. Moses, S. Castro, L. Kaelbling, T. Lozano-Perez, and N. Roy, "Active learning of abstract plan feasibility," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[41] 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)*, 2020, pp. 10 459–10 465.

[42] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning Neuro-symbolic skills for Bilevel planning," in *Conference on Robot Learning (CoRL)*, 2022.

[43] W. Yuan, C. Paxton, K. Desingh, and D. Fox, "SORNet: spatial object-centric representations for sequential manipulation," in *5th Annual Conference on Robot Learning*, 2021. [Online]. Available: https://openreview.net/forum?id=mOLu2rODIJF

[44] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021. [Online]. Available: https://doi.org/10.1177/0278364920987859

[45] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep Visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[46] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[47] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, 2016.

[48] L. Wang, X. Meng, Y. Xiang, and D. Fox, "Hierarchical policies for cluttered-scene grasping with latent plans," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2883–2890, 2022.

[49] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "A general task and motion planning framework for multiple manipulators," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 3168–3174.

[50] W. Thomason, M. P. Strub, and J. D. Gammell, "Task and motion informed trees (TMIT*): almost-surely asymptotically optimal integrated task and motion planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 370–11 377, 2022.

[51] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 639–646.

[52] M. Toussaint, J. Harris, J.-S. Ha, D. Driess, and W. Hönig, "Sequence-of-constraints MPC: reactive timing-optimal control of sequential manipulation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 753–13 760.

[53] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[54] S. Tyree, J. Tremblay, T. To, J. Cheng, T. Mosier, J. Smith, and S. Birchfield, "6-DoF pose estimation of household objects for robotic manipulation: an accessible dataset and benchmark," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 081–13 088.

[55] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *CoRL*, 2018.

[56] J. J. Kuffner and S. M. LaValle, "RRT-connect: an efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[57] Z. Kingston and L. E. Kavraki, "Robowflex: robot motion planning with MoveIt made easy," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, oct 2022, pp. 3108–3114.

[58] L. Shimanuki and B. Axelrod, "Hardness of motion planning with obstacle uncertainty in two dimensions," *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1151–1166, 2021. [Online]. Available: https://doi.org/10.1177/0278364921992787

[59] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on Octrees," *Autonomous Robots*, 2013, software available at https://octomap.github.io. [Online]. Available: https://octomap.github.io

[60] I. A. Sucan and S. Chitta, "Moveit!" 2013. [Online]. Available: http://moveit.ros.org

[61] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*. USA: American Association for Artificial Intelligence, 2002, p. 476–483.

[62] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, 06 2017.

[63] E. Olson, "AprilTag: a robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3400–3407.

[64] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[65] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "RViz: a toolkit for real domain data visualization," *Telecommunications Systems*, vol. 60, no. 2, p. 337–345, oct 2015. [Online]. Available: https://doi.org/10.1007/s11235-015-0034-5