Object-Centric Planning for Long-Horizon Robotic Manipulation and Navigation

by
Aidan Curtis
B.S., Rice University (2020)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Object-Centric Planning for Long-Horizon Robotic Manipulation and Navigation

by

Aidan Curtis

B.S., Rice University (2020)

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

A primary objective within the robotics research community is the development of robotic agents capable of executing long-horizon tasks within complex and novel environments. The sparse and factored nature of object-centric planning makes it a good candidate for the reasoning engine inside such an agent. However, several challenges remain under an object-centric planning framework. Challenges arise in areas such as efficiently grounding states with novel objects in cluttered environments, maintaining efficiency under large object sets, and safe exploration and manipulation in partially observable and nondeterministic environments. This thesis examines these limitations and proposes several strategies for solving them while maintaining the generalizability and flexibility of object-centric planning in long-horizon tasks.

Thesis Supervisor: Leslie P. Kaelbling
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Joshua B. Tenenbaum
Title: Professor of Brain and Cognitive Sciences

Acknowledgments

First, I would like to express my deepest gratitude to my advisors Leslie, Tomas, and Josh for their support, guidance, and valuable insights. I am beyond grateful for the time and effort they invested into my intellectual development.

Additionally, I'd like to thank my collaborators Caelan, Jose, Rohan, Tom, Xiaolin, and Yilun for their contributions to the work in this thesis along with my labmates Ferran, Gustavo, Jiayuan, Nishanth, Rachel, Sahit, Willie, and Yang for their camaraderie, support, and fruitful discussions.

Lastly, I'd like to thank my partner Jessica, parents Marci and Tom, sisters Brenna and Miranda, and aunt Kiki for their unwavering love, support, and words of encouragment throughout this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis outlines a comprehensive approach for creating generalizable robotic manipulation policies, capable of effectively and safely interacting with large collections of unfamiliar objects in partially observable environments. The general framework subscribed to in this thesis is that of task and motion planning [38], which combines the established tools of long-horizon symbolic planning with shorter-horizon manipulation primitives such as sample-based motion planning, trajectory optimization, and reinforcement learning.

The field of robotic manipulation has witnessed the emergence of several methods aimed at enhancing robots' adaptability and capability to handle complex tasks in dynamic environments. Among these, reinforcement learning (RL) and learning from demonstrations have gained considerable attention. RL enables robots to learn optimal policies through trial and error by interacting with their environment. RL has demonstrated success in various domains; however, its reliance on extensive exploration and convergence issues can limit its applicability in long-horizon manipulation scenarios. Learning from demonstrations (LfD) describes a set of approaches that leverage human-generated demonstrations to teach robots effectively. Some methods that fall into this category include behavior cloning [35], inverse reinforcement learning [32], offline reinforcement learning [74], and program synthesis [29]. While LfD has proven efficient when provided with high-quality demonstrations in narrow problem settings, its performance can be constrained by the quality and diversity of the demonstrations provided, thus potentially hindering its ability to generalize in certain long-horizon manipulation tasks.

In light of the limitations presented by RL and LfD, task and motion planning (TAMP) emerges as a promising alternative for addressing long-horizon manipulation challenges. TAMP offers the distinct advantage of being able to generalize to arbitrary goals and adapt to novel scenes, making it particularly suitable for tackling problems with extensive constraints. This flexibility stems from TAMP's capacity to integrate high-level task planning, which is responsible for determining the sequence of actions, with low-level motion planning, which focuses on the robot's specific movements. By combining these generalizable methods, TAMP provides a comprehensive solution that can efficiently handle complex robotic manipulation tasks in a wide range of environments.

Despite the numerous benefits offered by task and motion planning, this approach is not without its limitations. One of the primary constraints of TAMP is its assumption of a complete world model, which may not always be available or attainable in real-world scenarios. This reliance on a full model can make TAMP's performance sensitive to discrepancies between the assumed and actual environment states. Additionally, TAMP's effectiveness tends to diminish when faced with large object sets, as the complexity of the planning problem increases significantly, leading to prolonged computation times and potential scalability issues. Lastly, TAMP's fully deterministic nature inherently lacks the provision for exploration, specifically goal-driven exploration. This lack of exploration can result in suboptimal or overly conservative solutions under determinized abstract transition models. In this master's thesis, we will thoroughly examine these limitations and seek potential avenues to enhance TAMP's adaptability and performance in the context of robotic manipulation.

The outline of this thesis is as follows. In Chapter 3 we propose a system for long-horizon object manipulation from human-specified goals and visual input. Beyond visual grounding for object-centric planning, our system, which we call Manipulation with Zero Models (M0M), calculates object affordances and affordance-specific object models dynamically in order to limit expensive post-processing operations and work in an open-world capacity [20]. In Chapter 4 we discuss one of the limitations of such a system and of object-centric planning in general. Namely, how can we ensure planning efficiency with large object sets. First, we look at the case where a majority of objects are irrelevant to the task and propose a method called Planning with Learned Object Importance (PLOI) where we use a graph neural network to learn object importance scores from examples [97]. We then use those importance scores to speed up planning. Next, we investigate the case where the large object sets contain mainly relevant objects. In a system we call GenTAMP, we show that it is more efficient to learn an abstraction that enables planning in a numerically qualitative abstraction of the original problem [22]. In Chapter 5 we investigate another limitation of the M0M system, purposeful exploration. The M0M system is capable of passive and one-step exploration, but mobile manipulation environments often necessitate long-horizon planning for exploration that may involve manipulation and visibility reasoning subproblems. In this chapter we will discuss this class of problems termed visibility-aware navigation among movable obstacles and our Look and Manipulation Backchaining (LaMB) algorithm for solving them [21].

Chapter 2

Background

2.1 Symbolic Task Planning

Task planning, a longstanding focus in AI with applications often aimed at robotics, deals with object-centric planning in fully discrete domains [42, 65]. A task planning problem can be defined by a tuple $\Pi = \langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}, I, G \rangle$, where $\mathcal{P}$ is a finite set of properties, $\mathcal{A}$ is a finite set of object-parameterized actions, T is a (potentially stochastic) transition model, $\mathcal{O}$ is a finite set of objects, I is the initial state, and G is the goal [92].

A *property* in this context is a real-valued function on a tuple of *objects*. Predicates, a special case of properties with binary outputs, and object types as unary properties are included. A state represents the assignment of values to all possible applications of properties in $\mathcal{P}$ with objects in $\mathcal{O}$. A goal is an assignment of values to any subset of the ground properties, representing a set of states implicitly. We use $\mathcal{S}$ to denote the set of possible states and $\mathcal{G}$ for the set of possible goals over $\mathcal{P}$. The transition model T maps a state, ground action, and the next state to a probability, defining the dynamics of the environment.

Classical planning approaches, such as STRIPS and PDDL, utilize first-order logic for representing states, actions, and goal conditions, allowing for concise and expressive problem formulations. This makes it easier to design algorithms that efficiently search the state space for an optimal or near-optimal plan using search-based algorithms like A* or state-space planning algorithms like Graphplan [6], SATPlan [66], and heuristic search planners [11, 52]. The ultimate objective of task planning is to discover a sequence of actions that, when executed starting from an initial state, leads to a goal state that satisfies a set of goal conditions.

2.2 Motion Planning and Control

Motion planning algorithms attempt to find a collision-free path for a robotic system or an autonomous agent through a geometrically-defined environment, often represented as a configuration space C. The configuration space comprises a set of feasible configurations q that the robot or agent can attain, and is subdivided into free space C_{free}, where no obstacles are present, and obstacle space C_{obs}, where the agent must avoid collisions. The objective is to find a continuous path $\gamma : [0, 1] \to C_{free}$ that connects a start configuration q_s to a goal configuration q_g, with $\gamma(0) = q_s$ and $\gamma(1) = q_g$, and adheres to the given geometric constraints.

Various algorithms have been developed to tackle geometric motion planning problems, such as the Probabilistic Roadmap [67] and Rapidly-exploring Random Trees [72, 64]. These algorithms aim to efficiently explore the configuration space and construct a collision-free path between the start and goal configurations while considering the underlying geometric structure of the environment.

Similarly, control through trajectory optimization involves determining the most efficient path for a robotic system or an autonomous agent to follow while avoiding obstacles and respecting constraints. Given a start and a goal position, the objective is to find a feasible and optimal trajectory that connects these positions while considering the physical limitations of the system. The optimization process is typically carried out by minimizing a cost function $J(x, u)$, which depends on the state variables x and the control inputs u.

In this optimization problem, the constraints include the initial state $x(t_0) = x_0$, the final state $x(t_f) = x_f$, the system dynamics represented by the function $\dot{x}(t) = f(x(t), u(t), t)$, and the inequality constraints $g(x(t), u(t), t) \leq 0$ that ensure the trajectory stays within feasible regions and obeys physical constraints.

To solve this problem, various techniques can be employed. One common approach is to discretize the problem and convert it into a finite-dimensional optimization problem that can be solved using standard numerical methods like Sequential Quadratic Programming (SQP) [3, 33].

2.3 Task and Motion Planning

Task and motion planning (TAMP) is a powerful framework that combines symbolic task planning and motion planning/trajectory optimization. TAMP aims to generate high-level plans consisting of a sequence of actions and their corresponding motion trajectories while respecting the system's kinematic and dynamic constraints, as well as the environment's geometric and logical constraints. This integration enables the generation of executable plans that not only satisfy the high-level goals but also guarantee the feasibility of the resulting motion trajectories.

To solve TAMP problems, various approaches have been proposed, including hierarchical methods [60], iterative sampling methods [41, 16], and optimization-based methods [105, 106]. All of these approaches assume full state knowledge. Several extensions to TAMP-based systems that actively deal with uncertainty from realistic perception, including substantial occlusions, have also been demonstrated [58, 48, 40], including some that support open domains [107] where the set of objects in the world is not known a priori. The key extension is planning in belief space, over distributions of object poses and space occupancy. Belief-space TAMP-based systems engage in active information gathering, for example, through manipulating objects to observe the contents of uncertain space [61]. Existing belief-space TAMP-based systems have not addressed general shape uncertainty, however; they model only uncertainty over a small set of possible object types and over 4- or 6-dof poses.

2.4 Limitations

2.4.1 Perception

Techniques for perception in robotics have experienced significant advancements in recent years, primarily driven by the rapid progress in machine learning and computer vision [95]. Object classification, detection, and segmentation have become essential components of a robotic system's perceptual capabilities, enabling them to recognize, locate, and differentiate objects in their environment. The advent of deep learning techniques such as Convolutional Neural Networks (CNNs) has facilitated the development of highly accurate and efficient models, such as the widely adopted YOLO [88], Faster R-CNN [89], and Mask R-CNN [51] architectures. These models have not only improved the performance of robotic systems in a variety of applications, such as autonomous vehicles [7], drones [55], and manipulation tasks [75], but have also contributed to a better understanding of the challenges involved in designing robust perception modules for real-world scenarios.

The development of end-to-end trained visual robotic policies has emerged as a way of integrating these advancements into robot systems [34]. These policies are learned directly from raw visual inputs and optimize the robot's actions without explicitly modeling the underlying environment or objects [63]. While these approaches have shown promising results in specific tasks, such as grasping [113] or navigation [62], their generalizability remains limited. The primary reason for this limitation lies in the fact that end-to-end trained policies are highly reliant on the quality, quantity and diversity of the training data [104]. Furthermore, these policies often struggle to adapt to novel situations, as they lack an explicit representation of the environment that could facilitate transfer learning or adaptation to new scenarios [79].

To overcome these limitations, researchers have explored various techniques for integrating perception into a model-based planning framework [57]. This paradigm shift aims to leverage the strengths of both data-driven and model-based approaches, by combining the ability of deep learning techniques to extract meaningful representations from raw data with the structure and prior knowledge provided by the planning models [49]. Some notable efforts include the incorporation of semantic segmentation into SLAM (Simultaneous Localization and Mapping) algorithms [77], the development of differentiable physics models for improved object manipulation [13], and the fusion of probabilistic graphical models with deep learning architectures to better model uncertainty in the environment [36]. While these approaches are still in their early stages, they hold great promise in enhancing the robustness and generalizability of robotic perception, thereby enabling robots to operate more effectively in complex and dynamic environments.

2.4.2 Scaling to Many Objects

TAMP systems' dependency on a high-level discrete search across objects and relationships limits their scalability when dealing with large object sets. The challenge of planning for problem instances comprising numerous objects fuels the continued research in the domain of lifted planning [91, 18]. Lifted planners, operating in STRIPS-like domains, bypass the computationally intensive preprocessing stage that involves grounding actions over all objects. An alternative approach to lessen the grounding load is to streamline the planning problem through the formation of *abstractions* [23, 25, 54, 1]. In this thesis, We will describe two approaches that handle different components of this problem.

Our first approach, called PLOI [97], utilizes graph neural networks (GNNs) [93, 68, 2], an increasingly popular option for relational machine learning, with proven effectiveness in planning applications [109, 78, 94, 92]. In comparison to logical representations [83, 73, 27], GNNs provide the inherent capability to handle continuous object-level and relational attributes. We capitalize on this during our experimentation, presenting our findings within a simulated robotic context.

Our second approach, GenTAMP [22], is designed to tackle planning problems involving multiple objects that are relevant to the goal, or where irrelevant objects are difficult to distinguish from relevant ones. For example, a robot may need to retrieve a single nail from a pile of similar objects. To simplify the problem, GenTAMP learns abstractions that enable it to plan in a qualitative numerical bisimulation of the original planning space. This bisimulation is independent of the number of objects in the problem, and scales linearly with the number of necessary objects rather than exponentially. By employing this approach, GenTAMP can effectively address complex planning tasks that were previously intractable.

2.4.3 Exploration & Partial Observability

The exploration problem in robotics is a fundamental aspect of model-based planning, typically involving the robot interacting with unknown environments to gather data and build a world model. One such approach is frontier exploration, which guides robots towards the boundaries between explored and unexplored areas, or "frontiers", for systematic expansion of the known environment [112]. Another strategy is Informative Path Planning (IPP), which optimizes the robot's trajectory to collect the most informative data [15, 4]. While frontier exploration and IPP provide powerful tools for exploration, they often do not consider the visibility of the environment. Thus, visibility reasoning has emerged as an approach incorporating occlusions and field-of-view constraints in planning [37, 44].

Partial observability poses a significant challenge in robotic task and motion planning (TAMP). In an environment with partial observability, a robot only has access to a limited subset of the world's state at any given moment. This limitation necessitates the robot to maintain a belief state - a probability distribution over all possible states - and update it based on observations and actions [59]. Some existing TAMP approaches have attempted to plan in belief space [40, 61]. However challenges still remain, especially in cases of mobile-base navigation with severely limited visual scope and large open-world environment uncertainty.

Chapter 3

Long-Horizon Planning from Vision

In this chapter, we will discuss a possible strategy for overcoming the state grounding limitation in object-centric planning for robotics. More specifically, we will outline a system for task and motion planning from estimated affordances, which we call M0M. We leverage the planning capabilities of general-purpose task and motion planning (TAMP) systems [38]. These planners can construct long-horizon manipulation plans to achieve complex goals requiring tightly-knit discrete and geometric decision making. The principal realization underlying our methodology is that these planners do not necessarily demand a perfect and exhaustive world model, contrary to common assumptions. Instead, they only require responses to a series of "affordance queries", which can be addressed directly via perceptual data. These varying queries might employ diverse representations of entities, including point clouds, meshes, or image properties, depending on what is most suitable for the specific query.

Consider the example problem illustrated in figure 3-1. The objective is to have all objects positioned on a green designated area. Crucially, the robot does not have pre-existing geometric models of objects and lacks information regarding the presence and quantity of objects in the environment. It utilizes RGB-D images as input, which it segments and analyzes to identify potential surfaces and objects. Goals are communicated through first-order logical expressions.

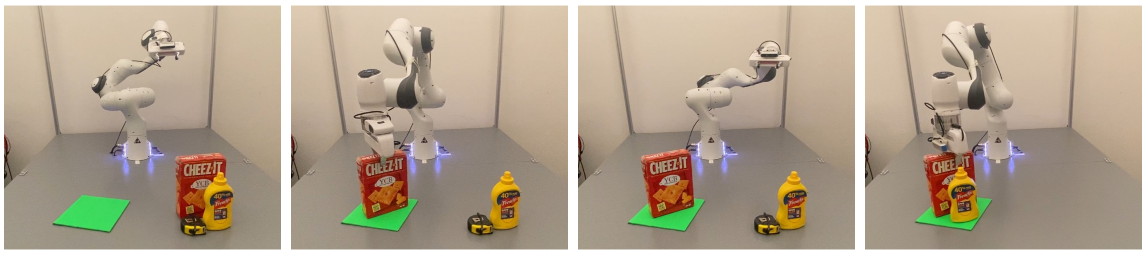

Figure 3-1: The robot starts with one object (the cracker box) visible. The goal is for all perceivable objects to be on a green target region. Multiple re-perceiving and re-planning steps are necessary to achieve the goal.

$$\forall obj.\ \exists region.\ \texttt{On}(obj, region) \wedge \texttt{Is}(region, \texttt{green}).$$

This equation incorporates a spatial relationship concerning object volumes, `On`, which the system can strategize to accomplish through actions like picking up and placing. It also includes perceptual characteristics like color (`green`), computed from input images by the system. Notably, the goal does not single out any objects by name. This is because, in the context of our problem, object instances do not have distinct identities and are identified through their properties. Consequently, goals broadly and specifically quantify over perceived objects, which can significantly vary in number and properties across or within problem instances.

In the beginning, a pair of objects are concealed behind the tall cracker box, rendering them invisible to the robot. Detecting only a single object on the table, the robot's initial action is to pick up and place the cracker box on the green designated area. Upon reevaluating the scene, the robot identifies two previously unseen objects. In order to satisfy the goal formula, the robot recalculates its strategy and purposefully relocates the cracker box to a temporary spot to create space for the tape measure and mustard bottle. Ultimately, the robot devises a new positioning strategy for the cracker box that avoids collisions with the other two objects, while simultaneously satisfying the goal.

This example demonstrates key traits of intelligent manipulation systems. Firstly,

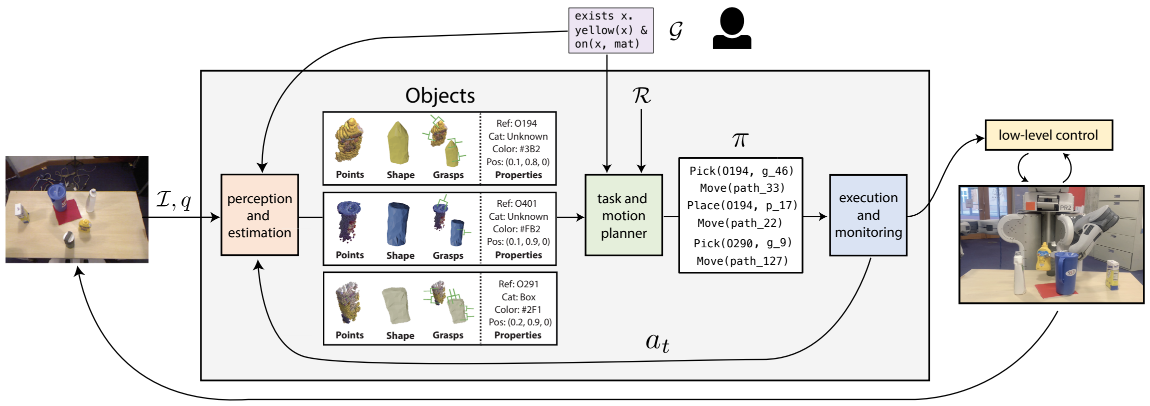

Figure 3-2: Structure of the goal-conditioned M0M policy, which maps RGB-D and robot configuration observations to robot position-controller commands.

our robot functions without needing to to explicitly identify specific object models or categories within its surroundings. In addition, unlike many methods that develop task-specific policies, our strategy can achieve any goal expressible in first-order logic using the provided relational vocabulary, without the need for images, physical demonstrations, or additional training for new tasks. Our approach considers the complex interplay between high-level planning and low-level geometry, such as obstruction, displaying reasoning that exceeds just lifting and placing objects in set positions. It formulates geometrically feasible plans under limited visibility, where objects may be partially or entirely unseen, but previously noted or included in the goal, combining shape completion, state estimation, and visibility reasoning. Lastly, our system integrates feedback during execution to reassess the plan when complications arise or more information becomes available. It maintains continuous memory throughout the re-planning process to enhance state estimation and manage scenarios where objects become hidden.

3.1 Method

Our method, termed Manipulation with Zero Models (M0M) [20], is the strategy we adopt. The M0M technique generates a "most likely" approximation of the world state, combining a segmentation of the existing scene and deductions drawn from previous observations, actions, and the objective. Following this, it computes a multi-step motion plan to fulfill the goal, based on that interpretation. It executes one or more steps of the plan, reassesses the scene, verifies if the goal has been met, and if not, reinitiates the planning process. This straightforward approach to dealing with partial observability proves to be unexpectedly effective in many scenarios, though it could be extended to include explicit belief-space planning [58, 40]. A system diagram illustrating the M0M manipulation policy to achieve a specified goal $\mathcal{G}$ can be seen in Figure 3-2.

The policy employs a set of parameterized manipulation actions $\mathcal{A}$ and modules which fall into one of three categories: robot-centric operations are coordinated through the $\mathcal{M}_{\mathcal{R}}$ modules, perceptual representations are computed by $\mathcal{M}_{\mathcal{P}}$ modules, and object-centric affordances are calculated by $\mathcal{M}_{\mathcal{A}}$ modules. Notably, the planner interacts with the physical world solely through these modules.

During each decision-making cycle, the robot captures the current RGB-D image I with its camera and obtains its current joint configuration q from its joint encoders. It then segments surface point clouds S and object point clouds O from each input RGB-D image. The estimated state is revised based on the segmented objects, surface point clouds, robot configuration, previous action, and the goal. This updated estimated state, along with the goal $\mathcal{G}$, actions $\mathcal{A}$, and modules $\mathcal{M}_R \cup \mathcal{M}_A$, constitute an implicit TAMP planning problem, where interaction with the state is restricted to calls to the specified modules. This is a highly adaptable strategy, applicable to a range of manipulation domains with different perception and affordance modules; we will elaborate on the modules for a specific implementation of M0M in the remainder of the paper.

If the goal $\mathcal{G}$ can be attained from the current state s, PLAN will yield a plan π, composed of a finite sequence of instances of the actions in $\mathcal{A}$. If the plan is void, the current state s meets the goal and the policy successfully concludes. If not, the robot performs the first action $\pi[0]$ utilizing its position controllers and repeats this procedure by reevaluating the scene.

3.1.1 PDDLStream formulation

M0M employs PDDLstream[39], a pre-existing, open-source, domain-agnostic planning framework designed for hybrid discrete-continuous domains. PDDLstream ingests models of manipulation actions, given as Planning Domain Definition Language (PDDL) operator descriptions (refer to Figure3-3), and a collection of samplers (also known as *streams*). These samplers generate potential values for continuous parameters, including joint configurations, grasps, object placements, and robot motion trajectories that meet the constraints specified in the problem's vocabulary (see Figure 3-4).

It's important to note that aside from a brief declaration of the properties their inputs and outputs adhere to, each stream's implementation is regarded as a blackbox. Consequently, PDDLstream remains impartial to the representation of stream inputs and outputs and to whether operations are engineered or learned from data. This flexibility allows the seamless integration of cutting-edge machine learning techniques, which can be utilized as-is during planning. The PDDLstream planning engine will then automatically combine them with other independent operations.

```
(:action move
 :parameters (?q1 ?t ?q2)
 :precondition (and (Motion ?q1 ?t ?q2) (HandEmpty) (AtConf ?q1)
                    (forall (?oc2 ?p2) (imply (AtPose ?oc2 ?p2)
                    (CFreeTrajPose ?t ?oc2 ?p2))))
 :effect (and (AtConf ?q2) (not (AtConf ?q1))))

(:action place
 :parameters (?q ?oc ?g ?p ?oc2 ?p2)
 :precondition (and (Grasp ?oc ?g) (Kin ?q ?g ?p) (Stable ?oc ?p ?oc2 ?p2)
                    (AtConf ?q) (AtGrasp ?oc ?g) (AtPose ?oc2 ?p2))
 :effect (and (HandEmpty) (AtPose ?oc ?p) (On ?oc ?oc2)
              (not (AtGrasp ?oc ?g))))
```

Figure 3-3: A PDDLstream description of `move` and `place` actions.

The PDDLstream planning language facilitates the description of problems that can be addressed by a range of PDDLstream planning algorithms [39]. The responsibility for querying perceptual operations (in the form of streams) rests with the PDDLstream planning engine, enabling it to autonomously determine online which operations pertain to the problem and the number of generated values required. Moreover, certain PDDLstream algorithms, such as the FOCUSED algorithm, employ a lazy querying approach to perceptual operations to circumvent unnecessary computation. Consequently, the planner doesn't execute resource-intensive perceptual operations on images and point clouds to predict properties and grasps unless the segmented object or property is deemed pertinent to the problem.

In PDDL, an action is characterized by a series of free parameters (`:parameters`), a precondition logical formula (`:precondition`) that must be satisfied for the action to be executed correctly, and an effect logical conjunction (`:effect`) that delineates state alterations upon the action's execution. Figure 3-3 presents the PDDL depiction of the `move` and `place` actions for M0M.

The `move` action represents the robot's collision-free motion when it's not holding anything. On the other hand, the `place` action encapsulates the immediate transition from the robot's hand exerting a force to hold an object to the cessation of that force and the subsequent release of the object.

A state achieves the goal if the goal formula is satisfied within it. Even goal specifications involving quantifiers can be conveniently and automatically encoded in a PDDL configuration using *axioms*, which are logical deduction rules [87, 103, 39]. An axiom, much like an action, has a similar precondition and effect structure but is automatically inferred at each state. Due to their resemblance to actions, axioms can be easily integrated into PDDL, empowering a planner to efficiently tackle complex goal conditions, like those found in M0M.

In addition to the domain and problem descriptions typical of `and` STRIPS planning formulations, PDDLstream includes stream descriptions, maintaining a syntax similar to PDDL operator descriptions. A stream is defined by a set of input parameters (`:inputs`), a logical formula that all valid input parameter values must fulfill (`:domain`), a list of output parameters (`:outputs`), and a logical conjunction that all valid input parameter values and produced output parameter values are guaranteed to comply with (`:certified`). Alongside each stream is a procedure that maps input parameter values to a potentially infinite series of output parameter values. Figure 3-

4 showcases some of the streams utilized in M0M, which will be elaborated in the subsequent section.

3.1.2 Streams

In the following, we outline the streams and the constraint predicates they certify. We emphasize the difference between streams that can be directly engineered and those that require at least some degree of learning. The engineered streams under consideration are robot-centric operations that can be executed using the robot's fully-observed URDF, which represents the robot's kinematics and geometry. For instance, the `inverse-kinematics` stream computes configurations `?q` that meet the kinematic constraint `(Kin ?q ?g ?p)` with grasp `?g` and pose `?p`, using tools like IKFast [24].

The `plan-motion` stream generates a continuous trajectory `?t` between configurations `?q1` and `?q2` that adheres to joint limits and avoids self-collisions, thereby certifying `(Motion ?q1 ?t ?q2)`. This stream can be directly implemented by any readily available motion planner, such as RRT-Connect [70].

The `predict-grasps` stream generates grasps `?g` for object cloud `?oc` that are predicted to remain stably in the robot's hand, certifying `(Grasp ?oc ?g)`. The `predict-placements` stream generates poses `?p1` for object cloud `?oc1` that are predicted to rest stably on object cloud `?oc2` when at pose `?p2`, certifying `(Stable ?oc1 ?p1 ?oc2 ?p2)`).

3.1.3 Manipulation policy

The core of the manipulation policy, which relies heavily on a fast symbolic planner, is detailed in Algorithm 1. The solution strategy of M0M is exhibited in a flowchart in Figure 3-2. The policy is built upon the set of manipulation actions $\mathcal{A}$ and the engineered streams $\mathcal{S}_E$. It also necessitates an implementation of the learned streams $\mathcal{S}_L$. The policy is designed around a specific robot $\mathcal{R}$ and a defined goal $\mathcal{G}$. To adapt the policy to a new robot $\mathcal{R}$, a URDF description of the robot's kinematics $\mathcal{R}$ and a position configuration controller for the robot are required.

```
(:stream predict-grasps
 :inputs (?oc)
 :domain (ObjectCloud ?oc)
 :outputs (?g)
 :certified (Grasp ?oc ?g))
(:stream inverse-kinematics
 :inputs (?oc ?g ?p)
 :domain (and (Grasp ?oc ?g) (Pose ?oc ?p))
 :outputs (?q)
 :certified (and (Kin ?q ?g ?p) (Conf ?q)))
(:stream plan-motion
 :inputs (?q1 ?q2)
 :domain (and (Conf ?q1) (Conf ?q2))
 :outputs (?t)
 :certified (and (Motion ?q1 ?t ?q2) (Traj ?t)))
(:stream predict-placements
 :inputs (?oc1 ?oc2 ?p2)
 :domain (and (ObjectCloud ?oc1) (Pose ?oc2 ?p2))
 :outputs (?p1)
 :certified (and (Stable ?oc1 ?p1 ?oc2 ?p2)
                 (Pose ?oc1 ?p1)))
(:stream predict-cfree
 :inputs (?t ?oc2 ?p2)
 :domain (and (Traj ?t) (Pose ?pc2 ?p2))
 :certified (CFreeTrajPose ?t ?oc2 ?p2))
(:stream detect-property
 :inputs (?oc ?pr)
 :domain (and (ObjectCloud ?oc) (Property ?pr))
 :certified (Is ?oc ?pr))
```

Figure 3-4: A PDDLstream description of the streams, which represent engineered and learned operations.

In each decision-making cycle, the robot captures the current RGB-D image I from its camera and gets its current joint configuration q from its joint encoders. From every input RGB-D image, it segments the table point clouds T and object point clouds O. The segmented object and table point clouds, coupled with the robot configuration, constitute the current PDDLstream state s of the world and robot. This present state, along with the goal $\mathcal{G}$, actions $\mathcal{A}$, and streams $\mathcal{S}_E \cup \mathcal{S}_L$, shapes a PDDLstream planning problem. This problem is solved by SOLVE-PDDLSTREAM, a procedure symbolizing a generic PDDLstream planning algorithm. In certain situations, such as when a critical attribute is not identified, SOLVE-PDDLSTREAM will return **None**, indicating that the goal $\mathcal{G}$ cannot be achieved from the current state s. If not, SOLVE-PDDLSTREAM will return a plan π, composed of a finite sequence of instances of the actions in $\mathcal{A}$. If the plan is empty (i.e., $\pi = [;]$), it means the current state s has been verified to meet the goal, and the policy terminates successfully. If not, the robot implements the first action $a_1 = \pi[0]$ using its position controllers and repeats this process by reobserving the scene. It is important to note that this control structure requires the robot to observe the scene to confirm goal achievement; otherwise, the robot could falsely claim success after executing an open-loop plan.

Algorithm 1 The M0M policy

```
Assume: $\mathcal{A} = \{\texttt{move}, \texttt{move-holding}, \texttt{pick}, \texttt{place}\}$
Assume: $\mathcal{S}_E = \{\texttt{inverse-kinematics}, \texttt{plan-motion}\}$
Require: $\mathcal{S}_L = \{\texttt{predict-grasps}, \texttt{predict-placements},$
    $\texttt{predict-cfree}, ..., \texttt{detect-attribute}\}$
1: procedure EXECUTE-POLICY($\mathcal{R}, \mathcal{G}$) Robot URDF $\mathcal{R}$, goal $\mathcal{G}$
2:     while True do
3:         $I, q \leftarrow$ OBSERVE()                          ▷ RGB-D image $I$, robot conf $q$
4:         $T, O \leftarrow$ SEGMENT($I$)                       ▷ Tables $T$, objects $O$
5:         $s \leftarrow$ OBJ-STATE($I, T, O$) $\cup$ ROBOT-STATE($\mathcal{R}, q$)
6:         $\pi \leftarrow$ SOLVE-PDDLSTREAM($s, \mathcal{G}, \mathcal{A}, \mathcal{S}_E \cup \mathcal{S}_L$)
7:         if $\pi$ = None then return False                 ▷ Failure: unreachable
8:         if $\pi = [\,]$ then return True                   ▷ Success: $s \in \mathcal{G}$
9:         EXECUTE-ACTION($\mathcal{R}, \pi[0]$)
```

3.1.4 Segmentation of objects and surfaces

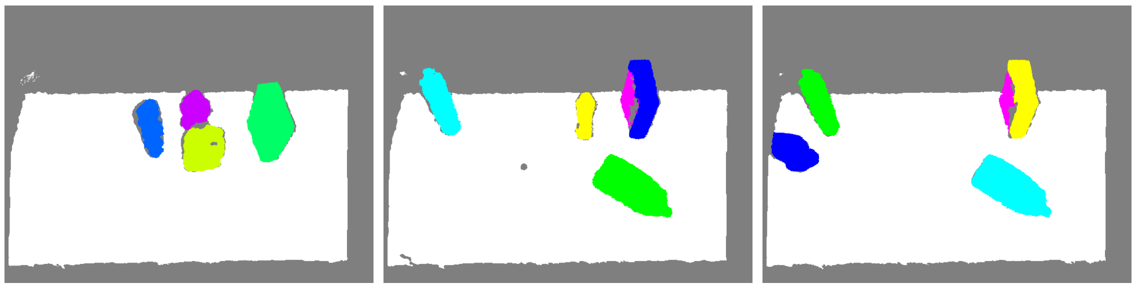

Figure 3-5: Three segmentation masks predicted by UOIS-net-3D during the system's execution. White pixels correspond to the table, chromatically-colored pixels correspond to object instances, and grey pixels are unassigned. Our system does not track objects over time, so each object instance is independently and arbitrarily assigned a color. *Left*: the initial segmentation mask. *Middle*: an intermediate segmentation mask after picking and placing two objects. *Right*: the final segmentation mask in a goal state.

We use category-agnostic segmentation to identify clusters of rigid points that move as a unified object upon manipulation. We examine three distinct segmentation methods: UOIS-net-3D, geometric clustering, and a hybrid approach.

The UOIS-net-3D method [111] employs a neural network model that processes RGB-D images to yield a segmentation of the scene. This model operates under the presumption that objects are typically situated on a table, thus it aims to identify and segment image regions corresponding to the table and the objects upon it.

Geometric clustering begins by eliminating points assigned to the table via UOIS-net-3D. It then uses the density-based spatial clustering of applications with noise (DBSCAN) [30] approach, which identifies linked components in a graph generated by connecting proximate points in the point cloud based on 3D Euclidean distance.

Our hybrid approach applies DBSCAN to the point cloud segmented by UOIS-net-3D in an effort to minimize under-segmentation. This is further complemented by post-processing to exclude degenerate clusters.

Figure 3-5 illustrates the segmentation mask as predicted by UOIS-net-3D. As observable in the figure, UOIS-net-3D generally segments the four instances accurately, though it does split the cracker box into two adjacent instances in the final two images.

We evaluated all three segmentation techniques on the ARID-20 subset of the Object Clutter Indoor Dataset (OCID)[102], and the GraspNet-1Billion[31] datasets. Comprehensive results are presented in Table 3.1. Each segmentation algorithm demonstrated strengths in different scenarios. In situations where objects possess simple shapes and are spread across the table, an Euclidean-based approach tends to deliver reliable predictions. However, in more cluttered environments, the learned approach often surpasses the Euclidean-based method. In complex scenarios where objects exhibit intricate geometries, while the performance of the learned method does decrease, it still outperforms the purely Euclidean-based approach. Across all experiments, the combined approach outperformed the purely learned one, which suggests the efficacy of applying DBSCAN and filtering to results predicted by neural networks.

Dataset	DBSCAN	UOIS-net-3D	COMBINED
OCID Uncluttered	**98.7**	95.5	97.2
OCID Cluttered	68.7	87.5	**89.6**
GraspNet-1Billion	69.1	80.1	**82.6**

Table 3.1: Comparison of the segmentation approaches in terms of F-measure.

3.1.5 Shape estimation

Shape estimation, a subroutine in our implementation of both the `predict-placements` and `predict-cfree` stream operations (Section 3.1.2), takes a partial point cloud as input and predicts a completed volumetric mesh. Here, we also explore a blend of neural-network-based and geometric methods.

The Morphing and Sampling Network (MSN) [76] is a neural network model that predicts a completed point cloud from an input of a partial one. Our geometric method enhances the partial point cloud by calculating the projection of the visible points onto the table plane. This heuristic is simple but effective, driven by the idea that the base of an object must be sufficiently large to stably support the visible part of the object. This becomes especially useful given a viewpoint that typically observes objects from above. For both methods, we incorporate a post-processing step that filters the result by back-projecting predicted points onto the depth image and removing any visible points that are closer to the camera than the observed depth value.

Mesh interpolation

Although it is feasible to use the estimated point cloud directly in downstream operations, such as treating the points as spheres or downsampling them into a voxel grid, it is often more precise and efficient to interpolate among the points to generate a volumetric mesh. The most straightforward way to accomplish this is by constructing the convex hull of the points. However, this method can significantly overestimate the volume when the object is non-convex and fail to find feasible plans when attempting to grasp non-convex objects like bowls. Thus, we generate the final volume by computing a "concave hull" in the form of an alpha shape [28], a non-convex generalization of a convex hull, from the amalgamation of the visible, network-predicted, and projected points. To facilitate efficient collision checking in the `predict-cfree` stream, we construct an additional representation that approximates the mesh as the union of several convex meshes, implemented by the Volumetric Hierarchical Approximate Convex Decomposition (V-HACD) [85].

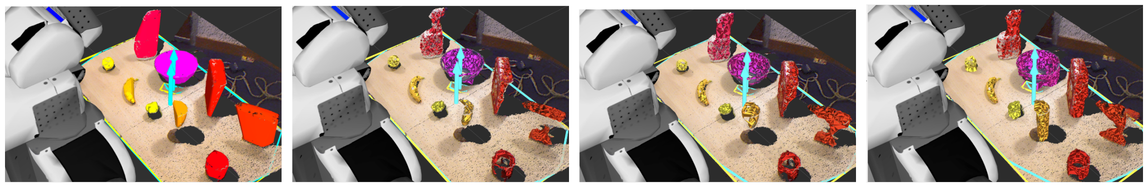

Figure 3-6: RViz visualizations of estimated shapes overlaid on top of raw point cloud data. a) the convex hull of the visible points only (V), b) the concave hull of the visible points only (V), c) the concave hull of the visibility-filtered visible and shape-completed points (VLF), and d) the concave hull of the visibility-filtered visible, shape-completed, and projected points points ($VLPF$).

Figure 3-6 illustrates the estimated meshes produced by four of the shape estimation strategies in a scene with a varied set of objects and no clutter. The first two images compare the mesh created by using the convex hull (Figure 3-6 left) versus a concave hull (Figure 3-6 middle-left) of the set of visible points (V). The convex hull can notably overestimate non-convex objects in certain regions, as seen with the spray bottle in the top left of the image and the power drill on the right side of the image. The last three images compare three strategies for populating the set of points to be used as the input to a concave hull. Including the shape-completed points from MSN (VLF) fills in some, but not all, of the occluded volume of each object, as demonstrated by the cracker box in the center of the image (Figure 3-6 middle-right). Also incorporating the projection of the points to the table ($VLPF$) more effectively fills in the occluded volume, but at the cost of overestimating the volume when the ground truth base projection is smaller than the visible base projection (Figure 3-6 right). We assessed the performance of these methods in four different domains, each on 2000 images taken from a randomly-sampled camera pose; detailed experiments and results are presented in table 3.2. The fully combined method ($VLPF$) generally performed the best across the domains and is the one we employ in the system experiments.

Clutter	Complex	*V*	*VP*	*VPF*	*VL*	*VLF*	*VLPF*
No	No	0.489	0.639	0.678	0.633	0.649	**0.703**
Yes	No	0.417	0.556	0.589	0.565	0.573	**0.612**
No	Yes	0.497	0.568	0.604	0.530	0.615	**0.620**
Yes	Yes	0.398	0.464	0.483	0.446	**0.503**	0.497

Table 3.2: Comparison of the shape-estimation strategies.

3.1.6 Grasp affordances

Grasp affordances refer to transformations between the robot's hand and an object's reference frame such that, if the robot positioned its hand at the specified pose and closed its fingers, it would secure the object in a stable grasp. These affordances are purely local and do not consider reachability, obstacles, or any other constraints.

The flexibility of our planning framework allows us to explore three different grasping methods for executing the `predict-grasps` stream, each requiring a partial point cloud as input. The Grasp Pose Detection (GPD)[47] initially creates grasp candidates by aligning one of the robot's fingers parallel to an estimated surface in the partial point cloud, and then scores these candidates using a convolutional neural network that has been trained on successful real-object grasps. GraspNet[82] employs a variational autoencoder (VAE) to learn a latent space of grasps which, when conditioned on a partial point cloud, generates grasps.

We have also devised a method, Estimated Mesh Antipodal (EMA), that conducts shape estimation utilizing the methods detailed in Section 3.1.5 and then pinpoints antipodal contact points on the estimated mesh. Specifically, to generate a new grasp, EMA samples two points on the surface of the estimated mesh that could potentially serve as contact points for the center of the robot's fingers. The pair of points is dismissed if the distance between them exceeds the gripper's maximum width, or if the surface normal at either of the corresponding faces is not roughly parallel to the line connecting the two points. Then, EMA samples a rotation for the gripper

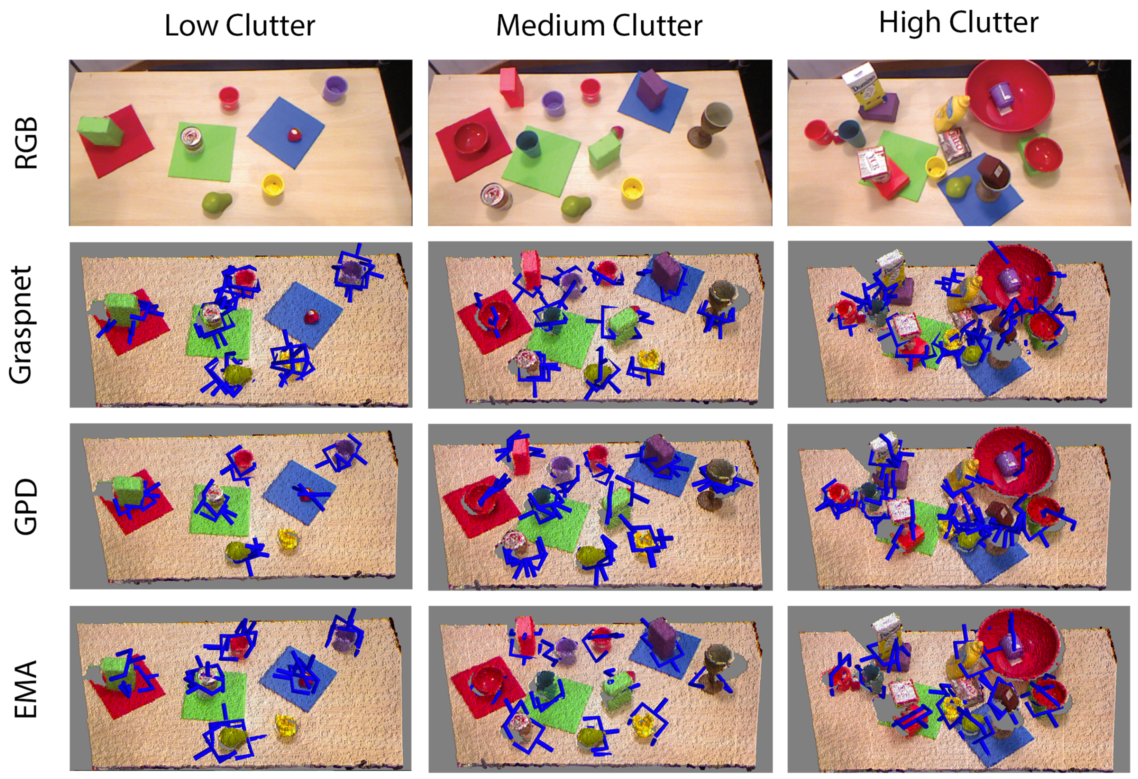

Figure 3-7: Grasps produced by each of the three grasp generation modules overlaid in green on the observed point cloud for three scenes with varying amounts of clutter.

around this line and yields the resulting grasp if the gripper, when open and at this transformation, does not collide with the estimated mesh. A key difference between EMA's and GPD's candidate grasp generation process is that EMA, through the use of shape estimation, is able to directly consider the occluded areas of an object rather than only the visible partial point cloud. Furthermore, it can account for potentially dangerous contacts between the robot's gripper and the object.

Figure 3-7 displays some of the grasps created by the aforementioned three techniques in three scenes with a range of clutter, where clutter provides extra occlusion opportunities. We carried out a real-world experiment to compare the success rates of GPD, GraspNet, and EMA. The specifics of the experiment and the outcomes can be found in Table 3-8. GPD and EMA surpassed GraspNet in our tests, in terms of both speed and precision, with EMA holding a slight advantage over GPD. For our system experiments, we opted for EMA.

Method	Uncluttered		Cluttered	
	Successes	Time	Successes	Time
GPD	27/35	8.7s	23/35	6.8s
GraspNet	21/35	48.3s	16/35	21.7s
EMA (*ours*)	27/35	10.3s	27/35	7.3s

Figure 3-8: Grasp success rates in uncluttered and cluttered settings.

3.1.7 Object properties

In our execution of the `detect-property` stream, we looked at detectors for two object properties: category and color. We employ Faster R-CNN [90], trained on the bowl-cup subset of IIT-AFF [86], to detect bowls and cups so the robot can determine which objects have the capacity to hold other objects. We also use Mask R-CNN [50], trained on real images from the Yale-CMU-Berkeley (YCB) Video Dataset [110] and a synthetic dataset we created using PyBullet [19], to classify any YCB objects [14] mentioned in the goal formula. In addition, we have simple modules that collate color statistics directly from segmented RGB images.

3.2 Experiments

3.2.1 Results

Lastly, we tested the entire M0M system's capability to solve difficult real-world manipulation tasks. For instance, Figure 3-9 presents a task where the goal is to place a mustard bottle on a blue target region:

$$\exists obj.; \exists region.; \texttt{On}(obj, region) \land \texttt{Is}(region, \texttt{blue}) \land \texttt{Is}(obj, \texttt{mustard}).$$

In the process of accomplishing this task, the robot moved two obstructing objects aside to securely pick up the mustard bottle and then position it on the goal region. Although not shown, the robot's initial attempt to grasp the mustard bottle was unsuccessful, leading the system to halt execution, re-observe, re-plan, and implement a new grasp that was successful this time.

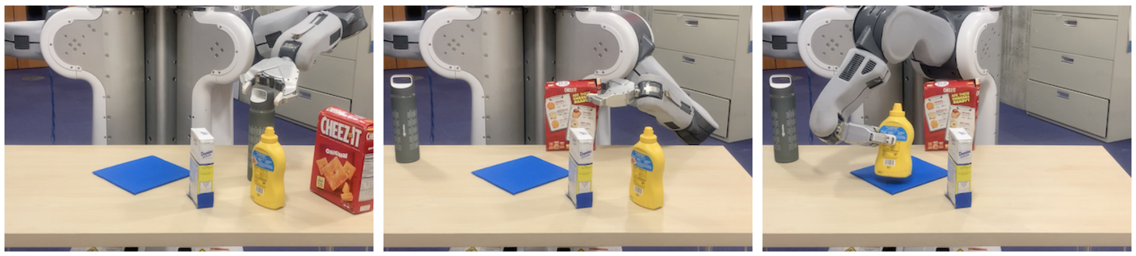

Figure 3-9: M0M acting to satisfy the goal for a mustard bottle to be on a blue target region. Left: for its first action, the robot picks up a water bottle that prevents the robot from safely reaching the mustard bottle. Middle: the robot also picks and relocates the obstructing cracker box. Right: finally, the robot places the mustard bottle on the blue target region.

3.2.2 Repeated trials

We carried out experiments encompassing five repeated trials in the real world for five different tasks, with the outcomes presented in Figure 3-10. In this context, tasks are loosely categorized as a collection of problems sharing the same goal formula and exhibiting similar initial states qualitatively. We provide a summary of the results here and delve into the task descriptions in the following subsections. In addition to these tasks, we tested the system on more than 25 separate problem instances.

The Iterations column signifies the average number of combined estimation and planning iterations executed per trial. Unless the initial state already fulfills the goal conditions, the system invariably undertakes two or more iterations because it must at least accomplish the goal and then confirm that the goal has indeed been met. In some cases, the system may carry out more than two iterations if the perception module discovers a new object due to undersegmentation, an action is aborted due to a failed grasp, or an action results in unexpected effects.

The Estimation, Planning, and Execution columns indicate the average time consumed per iteration in perception, planning, and execution, respectively. Each module was developed in Python to enable flexible support for multiple implementations of each module. Many perceptual operations involving raw point clouds could be accelerated by utilizing C++ instead of Python and running the system on graphics hardware. During the planning stage, most of the time is spent on collision checks,

Task	Iterations	Estimate	Plan	Execute	Success
1	2.0	29.6s	18.5s	16.1s	5/5
2	3.0	34.0s	37.4s	23.6s	5/5
3	3.0	36.8s	28.3s	40.5s	4/5
4	2.0	39.6s	41.6s	44.6s	5/5
5	2.4	47.7s	18.9s	28.3s	5/5

Figure 3-10: Full-system task completion experiments

especially when the robot is planning free-space movements. The overall runtime could be trimmed by concurrently planning motions for later actions while implementing earlier ones [40]. The Successes column denotes the number of times out of five trials that the system ended having confirmed that it achieved the goal. Our system managed to achieve the goal in every trial except for one, which was part of Task 3. These results demonstrate that this single system can perform a wide array of long-horizon manipulation tasks with robustness and reliability.

Chapter 4

Long-Horizon Planning with Large Object Sets

While the above approach serves as a good starting point for a robot system that plans to achieve goals in real-world settings from only visual input, several challenges still remain. One limitation of the task and motion planner used for M0M, and of object-centric planners in general, is that it struggles to scale with increasing object set sizes. Real-world settings contain thousands of objects, most of which are irrelevant and don't play a role in our reasoning about a specific task. If many objects do play a role in a specific task, it often makes more sense to reason about groups or numerical abstractions of those large object sets such "that pile" or "all but one". Below we will present two strategies that learn from data to create representations that lend themselves to efficient planning in the presence of large numbers of irrelevant and relevant objects.

4.1 Learning Object Importance

In this section, we suggest a method that learns to identify a subset of the entire object set that is needed for planning. For large problems with varying object counts, we train models that are agnostic the identity and quantity of objects. We adopt a convolutional graph neural network architecture [93, 68, 2] which can learn from a

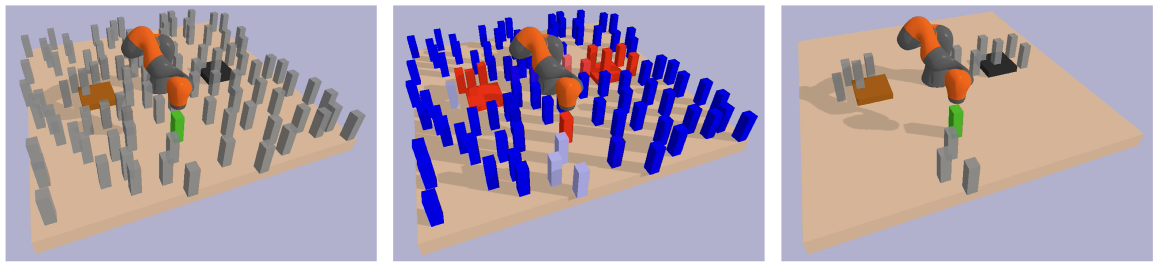

Figure 4-1: Example problem from the PyBullet domain. *Left:* The robot arm must move the target can (green) to the stove (black) and then to the sink (brown) while avoiding other cans (gray). *Middle:* GNN importance scores for this problem, scaled from blue (low importance) to red (high importance). We can see that cans surrounding the sink, stove, and target have been assigned higher importance score, meaning the GNN has reasoned about geometry. *Right:* The reduced problem in which the robot plans. Only objects with importance score above some threshold remain in the scene.

small set of training problems and be applied to more complex test problems with more objects.

Our strategy of predicting object importance has several advantages over alternative learning-based methodologies. First, it can operate with the planner and transition model as black boxes, and its runtime does not rely on the number of ground actions, given a constant number of objects. Additionally, it enables efficient inference, thus adding negligibly to the total planning time. Lastly, it tolerates a large margin of error in one direction, as the planning time can be significantly improved even if only some nonessential objects are excluded.

4.1.1 Methods

In this section, I'll illustrate our method for learning how to plan efficiently in large-scale problems. At a high level, we first learn a model capable of predicting a subset of the complete object set that is sufficient to solve the problem. During testing, we use the model to form a reduction of the planning problem, create a plan within this reduction, and then validate the derived plan in the original problem. In order to ensure completeness, we repeat this process, gradually expanding the subset until a solution emerges.

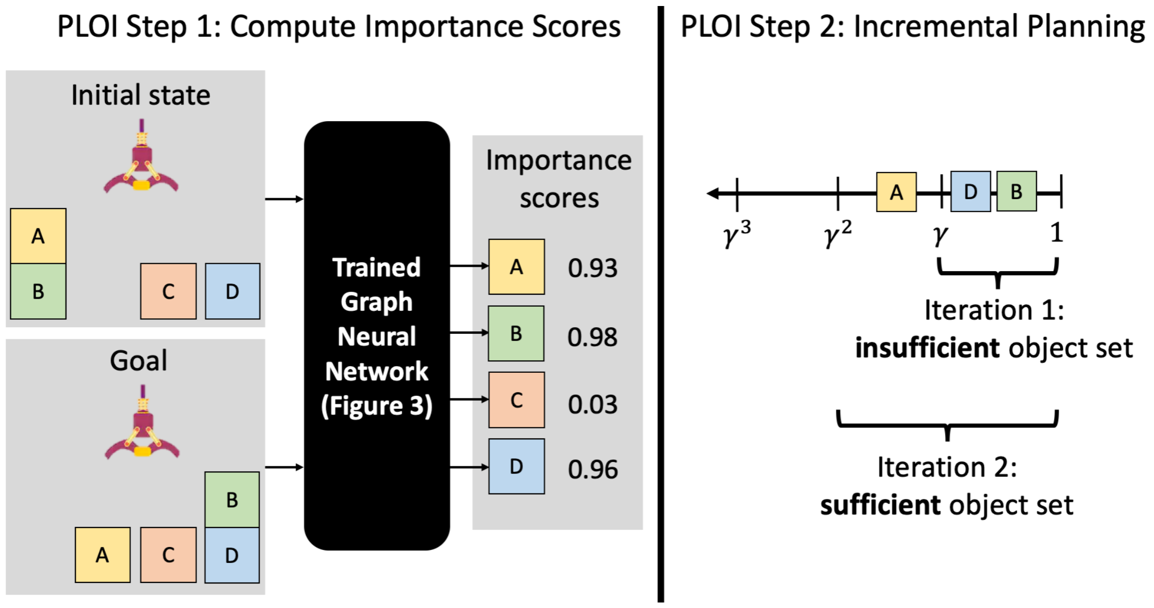
PLOI Step 1: Compute Importance Scores
Initial state
A
B
C
D
Goal
B
A
C
D
Trained Graph Neural Network (Figure 3)
Importance scores
A 0.93
B 0.98
C 0.03
D 0.96
PLOI Step 2: Incremental Planning
A
D
B
γ^3
γ^2
γ
1
Iteration 1: **insufficient** object set
Iteration 2: **sufficient** object set

Figure 4-2: Overview of our method, PLOI, with an example. *Left:* To solve this problem, the robot must move block A to the free space, then stack B onto D. The GNN computes the per-object importance score. Block C is irrelevant, and therefore it receives a low score of 0.03. *Right:* We perform incremental planning. In this example, $\gamma = 0.95$, so that $\gamma^2 \approx 0.9$. The first iteration tries planning with the object set {B, D}, which fails because it does not consider the obstructing A on top of B. The second iteration succeeds, because the object set {A, B, D} is sufficient for this problem.

We now present a more detailed explanation of PLOI, beginning with a formal description of the reduced planning problem. Given a planning problem $\Pi = \langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}, I, G \rangle$ and a subset of objects $\hat{\mathcal{O}} \subseteq \mathcal{O}$, the problem *reduction* $\hat{\Pi} = \textsc{ReduceProblem}(\Pi, \hat{\mathcal{O}})$ is defined by $\hat{\Pi} = \langle \mathcal{P}, \mathcal{A}, T, \hat{\mathcal{O}}, \hat{I}, \hat{G} \rangle$, where $\hat{I}$ (or $\hat{G}$) is I (or G) with properties only over $\hat{\mathcal{O}}$.

A *reduction* of the object set abstracts all components of the initial state and goal relating to the omitted objects, and restricts any ground actions that include these objects. This could result in a significantly simplified planning problem, but at the same time, it might oversimplify to the point where planning within the reduction produces an invalid solution or none at all. To segregate these sets from the beneficial ones we are seeking, we introduce the following definition.

Considering a planning problem $\Pi = \langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}, I, G \rangle$ and planner $\textsc{Plan}$, a subset of objects $\hat{\mathcal{O}} \subseteq \mathcal{O}$ is designated as *sufficient* if $\pi = \textsc{Plan}(\hat{\Pi})$ resolves Π, where

$\hat{\Pi} = \textsc{ReduceProblem}(\Pi, \hat{\mathcal{O}})$. In simpler terms, an object set is considered sufficient if planning in the associated reduction results in a valid solution for the original problem. An object set that lacks essential objects, like a required key to unlock a door or a necessary obstacle to avoid, will not be sufficient as planning will fail in the absence of the key, and validation will fail without the obstacle. Trivially, the complete set of objects $\mathcal{O}$ is always sufficient if the planning problem is satisfiable and the planner is complete. However, our goal is to identify a smaller sufficient set that allows for more efficient planning. Thus, our target is to learn a model that can predict such a set based on a given initial state and goal.

Scoring Object Importance Individually

Our objective is to learn a model that can recognize a small sufficient subset of objects provided an initial state, goal, and complete set of objects. Such a model has three fundamental requirements. First, given that our ultimate aim is to improve planning time, the model should allow for fast querying of object importance. Second, as we want to optimize the model using a moderate number of training problems, the model should allow for data-efficient learning. Lastly, as we desire to retain completeness when the original planner is complete, the model should provide some recourse when the first predicted subset turns out to be invalid.

These requirements exclude models that directly predict a subset of objects, as such models do not offer an apparent recourse when the predicted subset is insufficient. Furthermore, models that reason about sets of objects generally require vast amounts of training data and may consume considerable time during inference.

We instead choose to learn a model $f : \mathcal{O} \times \mathcal{S} \times \mathcal{G} \rightarrow (0, 1]$ that scores objects individually. The output of the model $f(o, I, G)$ can be interpreted as the probability that the object o will be included in a small sufficient set for the planning problem $\langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}, I, G \rangle$. We refer to this output score as the *importance* of an object. To get a candidate sufficient subset $\hat{\mathcal{O}}$ from such a model, we can simply take all objects with importance score above a threshold $0 < \gamma < 1$.

Using the graph neural network architecture, this inference process is highly efficient, requiring just a single inference pass. This parameterization also supports efficient learning, as the loss function decomposes as a sum over objects. As an optimization step, we always include in $\hat{\mathcal{O}}$ any objects mentioned in the goal, as such objects are a necessity in any sufficient set.

A significant benefit of individually predicting scores for objects is the natural recourse available when the initial candidate set $\hat{\mathcal{O}}$ does not lead to a solution: we can simply decrease the threshold γ and attempt again. In practice, we reduce the threshold geometrically (as described in Algorithm 2), ensuring completeness.

While predicting scores for individual objects simplifies the set prediction problem by constraining the hypothesis class, it is worth noting that this constraint also makes it impossible to predict certain object subsets. For instance, in planning problems where a specific number of identical objects are required, such as three eggs in a recipe or five nails for assembly, individual object scoring can only predict the same score for all copies. However, we find that this limitation is significantly outweighed by the benefits of efficient learning and inference in practical applications.

4.1.2 Training with Supervised Learning

We will now describe a method for learning an object scorer f given a collection of training problems $\mathbf{\Pi}_{\text{train}} = \Pi_1, \Pi_2, ..., \Pi_M$, with each $\Pi_i = \langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}_i, I_i, G_i \rangle$. The core concept involves framing the problem within the context of supervised learning. From each training problem Π_i, we aim to obtain input-output pairs $((o, I_i, G_i), y)$, where $o \in \mathcal{O}_i$ represents each object from the full set for the problem, and $y \in 0, 1$ is a binary label signifying whether o should be predicted for inclusion in the small sufficient set. Thus, the overall dataset for supervised learning will include an input-output pair for every object from each of the M training problems.

The y labels for the objects are not provided, and it can be quite challenging to accurately compute a minimal sufficient object set, even in smaller problem instances. We suggest a straightforward approximation method for automatically deriving the labels. Given a training problem Π_i, we execute a greedy search over object sets: starting with the full object set $\mathcal{O}_i$, we progressively remove a single object from the set, accepting the new set if it is sufficient, until no more individual objects can be eliminated without compromising sufficiency. All objects in the final sufficient set are labeled with $y = 1$, while the remaining objects are labeled with $y = 0$. This method, which requires planning several times per problem instance with full or near-full object sets to check sufficiency, leverages the fact that the training problems are substantially smaller and simpler than the test problems.

It's important to note that the aforementioned greedy method is an approximation, meaning that there could exist a smaller sufficient object set than the one produced. For example, consider a domain with a certain number of widgets where the only parameterized action is `destroy(?widget)`. If the goal is to end up with a number of widgets divisible by 10, and the full object set contains 10 widgets, the greedy procedure will end after the first iteration, as no object can be removed while retaining sufficiency. However, the empty set is actually sufficient as it leads to the empty plan, which trivially fulfills this goal. Despite such potential cases, this greedy method for deriving the training data is effective in practice at identifying small sufficient object sets.

Having prepared a dataset for supervised learning, we can advance in the standard fashion by defining a loss function and optimizing model parameters. To facilitate data-efficient learning, we employ a loss function that decomposes over objects:

$$\mathcal{L}(\mathbf{\Pi}_{\text{train}}) = \sum_{i=1}^{M} \sum_{o_j \in \mathcal{O}_i} \mathcal{L}_{\text{obj}}(y_{ij}, f(o_j, I_i, G_i)),$$

where y_{ij} signifies the binary label for the j^{th} object in the i^{th} training problem, and $f(o_j, I_i, G_i) \in (0, 1]$. We use a weighted binary cross-entropy loss for $\mathcal{L}_{\text{obj}}$, where the weight (10 in experiments) imposes a higher penalty on false negatives than false positives, in order to account for class imbalance.

Algorithm 2 Planning with Learned Object Importance

```
Require: Planning problem $\Pi = \langle \mathcal{P}, \mathcal{A}, T, \mathcal{O}, I, G \rangle$.
Require: Object scorer $f$.
    Hyperparameter: Geometric threshold $\gamma$.
    // Step 1: compute importance scores
 1: Compute score($o$) $= f(o, I, G)\ \ \forall o \in \mathcal{O}$
    // Step 2: incremental planning
 2: for $N = 1, 2, 3, ...$ do
        // Select objects above threshold
 3:     $\hat{\mathcal{O}} \leftarrow \{o : o \in \mathcal{O}, \text{score}(o) \geq \gamma^N\}$
        // Create reduced problem & plan
 4:     $\hat{\Pi} \leftarrow$ ReduceProblem($\Pi, \hat{\mathcal{O}}$)
 5:     $\pi \leftarrow$ Plan($\hat{\Pi}$)
        // Validate on original problem
 6:     if IsSolution($\pi, \Pi$) or $\hat{\mathcal{O}} = \mathcal{O}$ then
 7:         return $\pi$
```

4.1.3 Experiments

In this section, we present empirical results for PLOI and several baselines. We find that PLOI improves the speed of planning significantly over all these baselines.

Baselines

In our baselines, we evaluate a variety of experimental methods which span from the traditional planning approach to the most current learning-to-plan strategies. All GNN baselines undergo supervised learning using the plan set discovered by an optimal planner on minor training problems.

Pure planning applies the planner PLAN directly on the comprehensive test problems, including all objects. **Random object scoring** uses the incremental procedure, but in contrast to using a trained GNN for object importance scoring, each object is assigned a uniformly random importance score ranging from 0 to 1. This can be perceived as an ablation study, effectively removing the GNN from our process. **Neighbors** uses a simple heuristic that attempts incremental planning with all objects that are connected within a certain number of steps (denoted as L) in the relation graph to any goal-named object. This is applied for $L = 0, 1, 2, \ldots$. If no plan is found even after considering all goal-connected objects, we default to pure planning to maintain completeness. **Reactive policy** uses a learned goal-conditioned reactive policies, similar to many previous works [46, 92]. We adjust our GNN architecture to predict a ground action per timestep. The input remains identical, but the output now has a dual structure: one head predicts a probability over the set of actions $\mathcal{A}$, and the other predicts a probability over objects for every parameter of the action. During testing, we calculate all valid actions in each state and implement the one with the highest policy probability. This baseline does not utilize PLAN during testing. **ILP action grounding** replicates the method described by [43] using their best reported settings. We use the implementation provided by the authors for both training and testing, employing the SVR model with round robin queue ordering and incremental grounding with an increment of 500. Lastly, **GNN action grounding** Uses a GNN instead of the inductive logic programming (ILP) model used in the previous baseline. To achieve this, we adjust our GNN architecture to input a ground action along with the state and goal, and output the probability that the ground action should be incorporated into the planning problem.

Results

We evaluate on 9 domains: 6 classical planning, 2 probabilistic planning, and 1 simulated robotic task and motion planning. We selected several standard classical and probabilistic domains from the International Planning Competition (IPC) [12]. However, we altered these domains to generate problems involving a larger number of objects than usual. In each domain, we train on 40 problem instances and test on an additional 10 that are significantly larger, all using the PDDLGym library [96].

4.2 Learning Generalized Policies

While identifying minimal object sets can enhance efficiency in problems where most objects are irrelevant, it doesn't solve the scalability issues when plans involve many relevant objects or when one object is needed from a group of objects with similar attributes. This is where generalized planning comes into play.

	Pure Plan	PLOI (Ours)	Rand Score	Neighbors	Policy	ILP AG	GNN AG
Domains	Time	Time	Time	Time	Time	Time	Time
Blocks	7.47	**0.62**	49.99	**0.52**	7.25	2.33	52.95
Logistics	**8.55**	**6.44**	42.05	**15.40**	–	–	49.31
Miconic	87.71	**21.64**	–	93.86	–	–	–
Ferry	**12.64**	**7.52**	43.79	39.66	34.78	33.77	–
Gripper	24.48	**0.47**	56.58	37.63	28.94	5.71	86.29
Hanoi	**3.19**	**3.39**	**3.47**	4.63	–	6.15	7.55
Exploding	11.52	**0.81**	44.96	1.08	10.18	4.69	48.53
Tireworld	24.58	**4.38**	44.09	47.13	30.36	–	63.03
PyBullet	–	**2.05**	–	8.58	–	–	–

Table 4.1: On test problems, planning times in seconds over successful runs. All numbers report a mean across 10 random seeds, which randomizes both GNN training (if applicable) and testing. All times are in seconds; bolded times are within two standard deviations of best. AG = action grounding. Policy and AG baselines are not run for PyBullet because these methods cannot handle continuous actions. Across all domains, PLOI is consistently best and usually at least two standard deviations better than all other methods.

Generalized planners focus on identifying abstract high-level plans that are applicable to a broad, often infinite, class of problem instances sharing a common structure. For instance, a generalized plan such as "While there are objects in the box, pick up a reachable object in the box and place it on the table." can be applied to settings with any number of objects. Solving every problem instance individually with standard planning approaches can be highly inefficient because planners often slow down significantly as the solution length increases.

The aim of generalized planning is to bypass search in large state-space problem instances by detecting and utilizing structure in a few example problems with smaller state spaces. Moreover, languages used to express generalized plans often incorporate looping constructs, resulting in generalized plans that are more compact than traditional ones for larger domains. Discrete generalized planning methods have proven to accelerate planning in problems with large numbers of objects. However, this discreteness limits expressiveness on continuous features of the environment, such as the remaining capacity of a backpack, the water level of a camping thermos, or subtle collision constraints common in any robotic setting.

Although there are multiple approaches to generalized planning for classical AI, our method is the first to apply these strategies to continuous robotics domains where

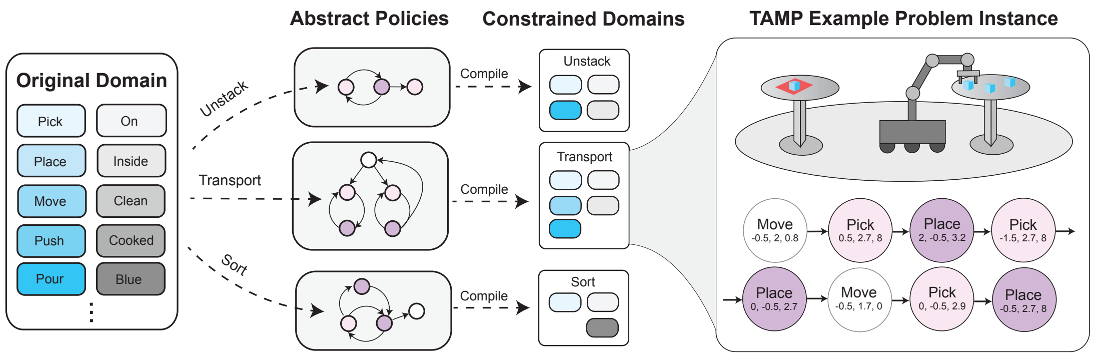

Figure 4-3: GenTAMP conceptual overview. Several robotics tasks are defined within a single domain. For each task, a policy is learned from a small set of example plans and compiled into a set of action and ordering constraints added to the original domain to create a task-specific domain. This task-specific domain forces TAMP search to follow the abstract policy, thus reducing the search space leading to faster planning.

the generalized plan necessitates complex continuous features.

We introduce GenTAMP, a novel framework for learning and executing generalized plans in environments with mixed discrete-continuous elements. Figure 4-3 provides a high-level overview of the GenTAMP approach. The detailed workings of this approach and its results will be discussed in the thesis.

4.2.1 Background

Generalized Planning

The generalized planning problem extends the task planning problem, introducing a meta-problem $Q = \langle \Pi_1, \Pi_2, \dots \rangle$ that comprises a (potentially infinite) set of task planning problems. The aim of generalized planning is to find a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ that, when consecutively applied from an initial state, creates a feasible sequence of actions reaching the goal for as many $\Pi_i \in Q$ as possible. Our problem formulation is based on the one proposed in [10], where problem instances do not share a state or action representation $\Pi_i = \langle \mathcal{S}_i, I_i, \mathcal{A}_i, G_i \rangle$.

Generalized TAMP Formulation

Different TAMP solvers rely on different TAMP formulations. We utilize the PDDLstream [41] problem definition language and associated algorithms as the foundation for generalized planning in our implementations. However, our approach remains compatible with other TAMP methodologies. This TAMP problem definition expands the entity set to incorporate *sampled entities* such as grasps, collision-free pose trajectories, and object placement samples. These entities are generated from high-dimensional nonlinear continuous spaces by *streams*, which are continuous samplers with both procedural and declarative elements. The procedural component is a function $g(\bar{o})$ that accepts a tuple of objects and produces sampled entities from a continuous space based on the inputs. The declarative component defines the semantics of these sampled entities and their relationship to existing entities in the domain through *certified predicates*. These predicates have the entities sampled by the stream as arguments and are referenced only in initial states, goals, and the preconditions of actions. Algorithms using the PDDLstream problem definition generally start with an insufficient set of sampled entities and certified predicates. During planning, they sample additional entities, appending corresponding certified predicates until a classical planner can achieve the goal.

A generalized plan for TAMP can be conceptualized as an abstract program designed to solve a particular problem. It defines the sequence and behaviour of high-level operations without directly specifying all of the continuous parameters. A step-by-step execution of this program, choosing parameters "on-the-fly" is not feasible due to the geometric and physical constraints imposed by the environment. For instance, in a packing task, if object placements are sampled and executed in a greedy manner, it might lead to situations where objects placed at the beginning of a plan obstruct the placement of objects later in the plan. Consequently, a generalized task and motion planner aims to construct a new domain constrained by each task. This constrained domain, when solved using a TAMP solver, accelerates the discovery of a ground plan in terms of actual computation time for any problem instance adhering

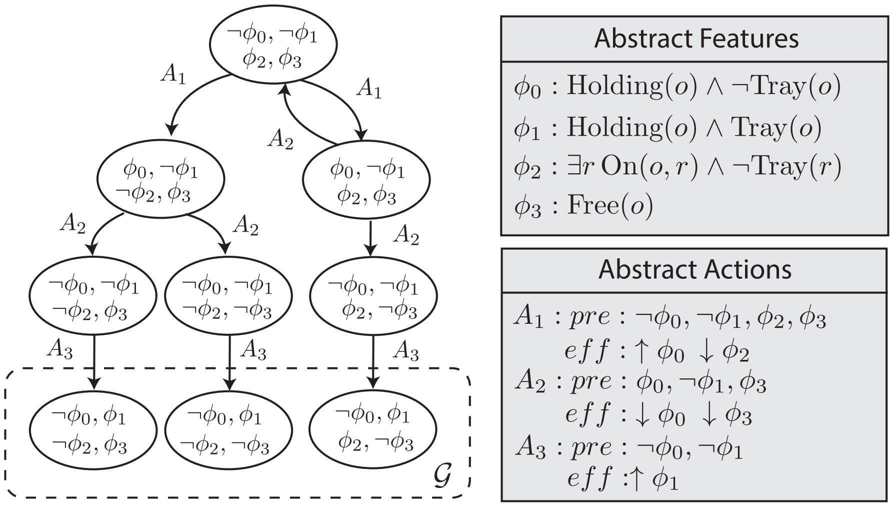
$\neg\phi_0, \neg\phi_1, \phi_2, \phi_3$
A_1
A_1
A_2
$\phi_0, \neg\phi_1, \neg\phi_2, \phi_3$
$\phi_0, \neg\phi_1, \phi_2, \phi_3$
A_2
A_2
A_2
$\neg\phi_0, \neg\phi_1, \neg\phi_2, \phi_3$
$\neg\phi_0, \neg\phi_1, \neg\phi_2, \neg\phi_3$
$\neg\phi_0, \neg\phi_1, \phi_2, \neg\phi_3$
A_3
A_3
A_3
$\neg\phi_0, \phi_1, \neg\phi_2, \phi_3$
$\neg\phi_0, \phi_1, \neg\phi_2, \neg\phi_3$
$\neg\phi_0, \phi_1, \phi_2, \neg\phi_3$
$\mathcal{G}$
Abstract Features
ϕ_0 : Holding(o) $\wedge$ $\neg$Tray(o)
ϕ_1 : Holding(o) $\wedge$ Tray(o)
ϕ_2 : $\exists r$ On(o, r) $\wedge$ $\neg$Tray(r)
ϕ_3 : Free(o)
Abstract Actions
$A_1 : pre : \neg\phi_0, \neg\phi_1, \phi_2, \phi_3$
$eff : \uparrow \phi_0 \downarrow \phi_2$
$A_2 : pre : \phi_0, \neg\phi_1, \phi_3$
$eff : \downarrow \phi_0 \downarrow \phi_3$
$A_3 : pre : \neg\phi_0, \neg\phi_1$
$eff : \uparrow \phi_1$

Figure 4-4: An abstract policy for the working LOAD task example. The generalized plan that solves this LOAD task picks an object (A_1) and places it on the tray (A_2) until either the tray is full ($\neg\phi_3$) or no objects are remaining on the table ($\neg\phi_2$). After one of these two features evaluates to false, the tray is grasped (A_3). Here `Free`(o) is used as a shorthand for the larger feature $\exists p, r \forall o_2, p_2$: `Tray`$(r) \wedge \neg$`On`$(o, p, r) \wedge$ `On`$(o_2, p_2, r) \Rightarrow \neg$`CFree`$(o, p, o_2, p_2, r)$

to the task specification. (Refer Figure 4-3)

This formulation presents a notable departure from traditional generalized planning in several critical ways. Firstly, in line with TAMP robotics applications, all tasks are defined under a single lifted domain comprising the same set of actions, streams, and properties that describe the world. Secondly, goals and initial states are now expressed in a logical language that incorporates continuous geometric and physical parameters (for example, a pose being within a specific region). Lastly, due to the impracticality of direct execution of a generalized TAMP plan, we instead measure the speed of search across numerous test problem instances.

Working example.

To build an understanding of generalized TAMP, we illustrate a basic working example of a generalized TAMP problem, adhering to the specification discussed above. We start by outlining a robotics domain that includes a free-flying robotic gripper agent capable of picking up and placing movable objects.

The domain includes predicates such as `Holding`(o), `On`(o_1, p, o_2), `Movable`(o), `Tray`(o), and `HandEmpty`. It also incorporates certified predicates like `Pose`(o, p, s), defining possible object placement, and `CFree`(o_1, o_2, p_1, p_2), indicating the non-collision of two objects at particular poses on the same surface. The domain further includes the following action schemas:

```
Pick($o,p,s$)
 pre: HandEmpty, On($o,p,s$), Pose($o,p,s$)
 eff: Holding($o$), ¬On($o,p,s$), ¬HandEmpty
Place($o,p,s$)
 pre: Holding($o$), Pose($o,p,s$), $\forall o_2, p_2$ : CFree($o,p,o_2,p_2$)
 eff: ¬Holding($o$), On($o,p,s$), HandEmpty
```

The domain also contains two streams that produce sampled entities and certified predicates. The first stream, `SamplePlacement`(o, s), accepts a surface and an object, samples a pose entity p, and appends `Pose`(o, p, r) to the initial state. The second stream, `CheckCollision`(o_1, p_1, o_2, p_2), examines for collisions and adds the certified predicate `CFree`(o_1, p_1, o_2, p_2) if no collision is detected.

We then define the LOAD task, one among several potential tasks within this domain. The task's objective (G_{LOAD}) is to ensure all objects are on the tray or that the tray is filled, with the initial state (I_{LOAD}) being all objects placed on the table. The LOAD task comprises problem instances $P_N, \ldots$, where N represents the number of objects. Each problem within this task begins with an initial entity set comprising a robotic gripper agent r, a set of N objects $b_1, \ldots b_N$ and their initial poses $bp_1, \ldots bp_N$, a tray t with an initial pose tp, and a table src. The literals in a problem instance's initial state are `HandEmpty`, `Tray`(t), ,`On`(t, tp, src), ,`Pose`(t, tp, src), `On`(b_i, bp_i, src), `Pose`(b_i, bp_i, src), and `Movable`(b_i) for all b_i. The optimal policy for this straightforward task would be to repeatedly pick an object and place it on the tray until it is full, followed by picking up the tray. This task is a subtask of the TRANSPORT task, which is elaborated upon and evaluated in the Experiments section.

4.2.2 Methods

GenTAMP Training

In this section, we outline our suggested approach for learning a deterministic policy from a small number of TAMP example plans. Initially, we leverage these example plans to extract a set of unique state features within the domain. These features are then combined with a range of constraints to form a SAT theory. Solving this theory yields a subset of features from the original pool, collectively forming an approximately valid abstraction. We utilize these selected features along with an initial state to construct a fully observable non-deterministic (FOND) planning problem. The solution to this FOND problem manifests as a policy, which when queried with an evaluation of the selected features, deterministically provides an abstract action. The specifics of the policy learning pipeline are provided in Algorithm 1. This strategy for generalized planning was first presented by [8]. In subsequent sections, we present a novel adaptation of this framework, suitable for problem domains with continuous state and action spaces.

Algorithm 3 GenTAMP Training

```
Require: $P = \{P_1, P_2, \dots, P_{N-1}\} \subseteq Q$
Ensure: $\pi_\phi$
 1: $\mathcal{D} \leftarrow \{\}$, pool $\leftarrow \{\}$
 2: for $P_i \in P$ do
 3:     $\mathcal{D} \leftarrow \mathcal{D} \cup$ TAMP($P_i$)
 4: for $c \in \{1 \dots$ max_complexity$\}$ do
 5:     pool $\leftarrow$ Prune(pool $\cup$ Generate($pool$))
 6: $\mathcal{T} \leftarrow$ EncodeSAT($\mathcal{D}$, pool)
 7: $F, \bar{A} \leftarrow$ SATSolve($\mathcal{T}$)
 8: $\bar{I}, \bar{G} \leftarrow F(P)$
 9: $\pi_\phi \leftarrow$ FONDPlan($F, \bar{A}, \bar{I}, \bar{G}$)
10: return $\pi_\phi$, F
```

State and action abstractions

In order to establish a policy with inputs that comply with a standard representation across problem instances within a task, we initiate a transformation of the problem-instance state space. This is achieved using a set of Boolean features $\phi_b : \mathcal{S}_i \rightarrow \{0, 1\}$ applicable to any instance state space $\mathcal{S}_i$, and numeric features $\phi_n : \mathcal{S}_i \rightarrow \mathbb{N}_0$, along with their corresponding qualitative projections $\phi\tilde{n} : \mathcal{S}_i \rightarrow \{0, >0\}$. These features, defined as relations over entity sets, can be applied to states with varying quantities of entities and properties. For instance, the numeric feature $\phi_n(s) \equiv |\{x \in \mathcal{O}_s : \texttt{Blue}(x)\}|$ receives a state s and returns the count of blue objects within that state. This feature is applicable to any state, independent of the number of objects present.

Typically, these features are inadequate for reconstructing the state, and hence provide an abstract state representation $\phi(s)$, which is an evaluation of each ϕ_b, $\phi_{\tilde{n}}$ for a state from any of the problem instances in the generalized planning problem. This abstract transformation can be applied to the initial and goal states of a problem instance to derive an abstract initial state $\bar{I}$ and goal state $\bar{G}$. Abstract actions $\bar{A}$ are characterized by preconditions and effects that are Boolean features or qualitative evaluations of numeric features. Abstract actions can be easily inferred from the features ϕ_b, $\phi_{\tilde{n}}$. More specifically, the effects of a derived abstract action are the qualitative effects over the features, and the preconditions of a derived action are the features that qualitatively align in the states where that qualitative effect was observed. These abstract actions are no longer deterministic as actions that decrease a numeric feature can result in two possible effects on the qualitative abstract state: reducing the feature value to 0 or maintaining it above 0.

An abstraction for a task is denoted by the tuple $\langle F, \bar{A}, \bar{I}, \bar{G} \rangle$, where F denotes the set of Boolean and qualitative numeric features. Given such an abstraction, it is possible to determine an abstract policy $\pi_\phi(s)$ either by using generate-and-test methods [100], or more efficiently, through the application of a non-deterministic planner [84]. This abstract policy can then be applied to novel problem instances, irrespective of the number of objects they contain.

The aim of our research is to learn ϕ_b and ϕ_n, as well as the necessary abstract actions, for a task Q using data gathered from transitions sampled from the problem instances of task Q. It is essential that the learned feature sets and the resulting abstract actions adhere to three crucial properties: soundness, completeness, and goal-distinguishability.

These three abstraction properties are proven to be *valid*, meaning they sufficient to guarantee that a policy within the space of abstract states and actions can always be refined to a concrete plan in any domain problem instance [9].

Collecting training data.

To generate valid abstractions, a comprehensive search through the state space is required in order to confirm soundness, completeness, and goal distinguishability on every transition. However, due to the continuous and high-dimensional nature of TAMP problems, it is not feasible to enumerate every attainable state and transition in the environment. Hence, we concentrate on constructing *approximately valid* abstractions from a subset of potential transitions in the environment. Our strategy uses a TAMP solver to find example plans for each training problem instance, using the transitions from these plans to create approximately valid abstractions. The states that make up the plan encompass all symbolic predicates and entities, along with any sampled entities and certified predicates found in the plan's preimage.

Creating a feature pool

In order to select a set of features that form an approximately valid abstraction, we generate a pool of candidate features with bounded complexity and select a subset from these that collectively yield a valid abstraction. To enable to high-arity certified predicates with sampled entities, we devise a generative grammar that allows for features containing implications and a mix of existential and universal quantifiers.

A TAMP domain is accompanied by a set of primitive predicates $p(\bar{o})$. We also define a set of named variables $x_1, \dots, x_M$ where M represents the maximum predicate arity. These predicates and named variables serve as terminals in constructing a set of concepts $C(\bar{z})$ through primitive negation, conjunction, and a single implication. Concepts are merged to form quantified concepts $C_q(x, \bar{z}_1, \bar{z}_2)$, where x is a free variable, $\bar{z}_1$ is a tuple of bound variables, and $\bar{z}_2$ is a tuple of variables that are yet to be bound. Formulas containing only a single free variable are created by quantifying all but one of the free variables using chained universal and existential quantifiers. Lastly, features are derived from the quantified concepts that contain only a single free variable. These resulting features are functions that, given a state, return a natural number indicating the quantity of entities that can be plugged into the remaining free variable such that the feature's concept holds. The particulars of this generative grammar can be found in Table 4.2.

Each feature in this grammar is paired with a *feature complexity* score, which is defined as the maximum of the number of generative grammar rules used to create the feature and the number of arguments of the feature. This grammar is implemented in a bottom-up manner, combining primitives, concepts, and quantified concepts to form a complete feature, and discarding features when they surpass a fixed complexity limit (5 in our experiments).

Arguments	**Features**
$z_m \to x_1 \mid x_2 \mid \ldots \mid x_M$	$\phi_n \to f : s \mapsto \lvert\{x \in \mathcal{O}_s : C_q(x, \bar{z}, \emptyset)\}\rvert$
Concepts	**Quantified Concepts**
$C_r(\bar{z}) \to p(\bar{z})$	$C_q(x, \emptyset, \bar{z}) \to C(\{x\} \circ \bar{z})$
$C_r(\bar{z}) \to \neg\, p(\bar{z})$	$C_q(x, \bar{q} \circ \{z_1\}, \bar{z}_{2:N}) \to \exists z_1 . C_q(x, \bar{q}, \bar{z}_{1:N})$
$C_r(\bar{z}_1 \circ \bar{z}_2) \to C_{r_1}(\bar{z}_1) \wedge C_{r_2}(\bar{z}_2)$	$C_q(x, \bar{q} \circ \{z_1\}, \bar{z}_{2:N}) \to \forall z_1 . C_q(x, \bar{q}, \bar{z}_{1:N})$
$C \to C_r(\bar{z})$	
$C \to C_{r_1}(\bar{z}_1) \Rightarrow C_{r_2}(\bar{z}_2)$	

Table 4.2: Generative grammar for features that take in a state and generate per-entity first-order logical expressions.

This grammar specification embodies two key properties that are indispensable for TAMP domains. Initially, we employ arbitrary-arity features as opposed to binary features to cater to the high-arity predicates prevalent in most TAMP domains. To manage these higher arity predicates, our grammar integrates alternating quantifiers evaluated in Prolog [17]. Secondly, we incorporate certified predicates and sampled entities into the grammar via the terminals (p), so the resulting features blend geometric and symbolic properties. These modifications significantly augment the total number of features, necessitating more stringent pruning at every generation stage. We apply pruning of concepts and quantified concepts at each stage through *uniqueness* and *equivalence*. Uniqueness pruning examines each gathered transition to ensure that no feature of lower complexity shares the same value on every transition. Equivalence pruning employs standard logical inference rules to assess equivalence between logical formulas.

Outlined below are some representative quantified concepts from our operational LOAD task, including quantified concepts composed from both symbolic and geometric primitives from the TAMP specification.

- $f_1(o) \equiv \texttt{Holding}(o) \ \wedge \ \neg\texttt{Tray}(o)$
- $f_2(o) \equiv \texttt{Holding}(o) \ \wedge \ \texttt{Tray}(o)$
- $f_3(o) \equiv \exists p, r : \texttt{On}(o, p, r) \wedge \neg\texttt{Tray}(r)$
- $f_4(o) \equiv \exists p, r \forall o_2, p_2 : \texttt{Tray}(r) \wedge \neg\texttt{On}(o, p, r) \wedge \texttt{On}(o_2, p_2, r) \Rightarrow \neg\texttt{CFree}(o, p, o_2, p_2, r)$

Upon conversion of these quantified concepts to features, we obtain numeric evaluations of each concept. As an illustration, the feature corresponding to the final concept gives the count of objects that can be positioned on a tray without clashing with other objects already on the tray. This feature merges sampled continuous properties of the state, such as collision constraints, with symbolic properties like `On` and `Tray`. It's easy to see how this feature could be beneficial in a generalized task and motion plan for the LOAD task.

Abstraction Learning.

With a feature pool generated by the aforementioned grammar and the set of gathered transitions, our next step is to identify a subset from that feature pool that meets the soundness, completeness, and goal-distinguishability criteria for the example plans in the training dataset. We translate the desired constraints (explained in the appendix) into a SAT theory, then employ a SAT solver to create an approximately valid abstraction. Our methodology uses the OpenWBO [80] weighted Max-SAT solver and assigns weights to the features that correspond to the complexity of that feature in the grammar. When we apply the SAT solver to our operational LOAD task to enforce our criteria over the set of selected features, we discover that the set of features with the lowest weight that fulfill the correctness criteria is exactly those outlined in the previous section; the abstracted actions are in Figure4-4. This step is usually the most time-consuming part of the policy learning pipeline as the construction of the SAT theory requires evaluation of every numeric feature on all example transitions.

Due to the non-closed world assumption of sampled entities and certified predicates, we cannot assure the accuracy of any feature evaluation on the state that quantifies over continuous state variables. For instance, we cannot confirm the correctness of the Boolean feature "All object grasps are not reachable" because there is an infinite number of possible grasps. However, provided that the sampled entities are drawn uniformly from their respective continuous spaces, the features will be evaluated correctly in the limit of infinite samples. Given that this uniform assumption is violated when extracting sampled entities in the preimage of TAMP plans, we incorporate additional stream samples in each plan prior to abstraction learning.

Qualitative Numeric Planning

Given an abstraction, we implement the feature transformation to the initial and final states of the training instances to derive abstract representations of our initial state and goal. We subsequently pass this abstract fully observable non-deterministic planning (FOND) problem to a standard non-deterministic planner to derive a generalized plan. In this study, we employ the state-of-the-art PRP [84] as our FOND planner. Figure 4-4 illustrates the consequent generalized plan for the active LOAD task using our learned features.

GenTAMP Constraint Satisfaction

We will now explore how the acquired generalized plan can be concretized (or executed) in a fresh TAMP problem instance within the same task. Conventional approaches to generalized planning implement learned policies by randomly choosing from a set of ground actions that 1.) are feasible in the state and 2.) incite the same effects on the abstract state as the abstract action from the policy. However, the direct execution of a generalized task and motion plan is unfeasible due to additional geometric and physical constraints imposed by the environment.

A common structure for TAMP planners is an outer loop that iterates over plan "skeletons", which consist of operator instances with the discrete parameters bound, yet the continuous parameters are still free, and a set of constraints that must be met to validate the plan. An inner loop then searches for a satisfying assignment of the continuous parameters. To incorporate our learned policy into the standard TAMP framework, we create a new set of planning operators that encode the action-ordering constraints and abstract action preconditions and effects of the generalized plan, and input these new operators into PDDLstream. The resulting constrained domain can conduct a more efficient concurrent search through the joint space of skeletons and continuous-parameter bindings, thus diminishing the overall search time of the TAMP solver.

Taking our LOAD task as an example, a conventional TAMP solver would explore actions that place objects on the table. However, this would be strictly unfeasible in the constrained task-specific problem specification because the abstract LOAD policy stipulates that if an object is being held, the next action must augment the number of objects on the tray (Figure 4-4).

4.2.3 Results

We demonstrate GENTAMP on four continuous robotic TAMP tasks detailed below and demonstrated in Figure 4-5. We selected tasks that are physical incarnations of widely employed tasks in the generalized planning literature [99, 8, 56] or classically challenging TAMP domains with large numbers of objects. We outline the goal of each of the tasks and the qualitative behavior of the policies uncovered by GENTAMP.

We compare **GENTAMP** with two baseline methods. **Unguided TAMP** simply executes a state-of-the-art task and motion planner (the same one utilized for data collection). **Partial skeleton** employs the same task and motion planner but is supplied with an input skeleton that contains a known feasible action schema ordering

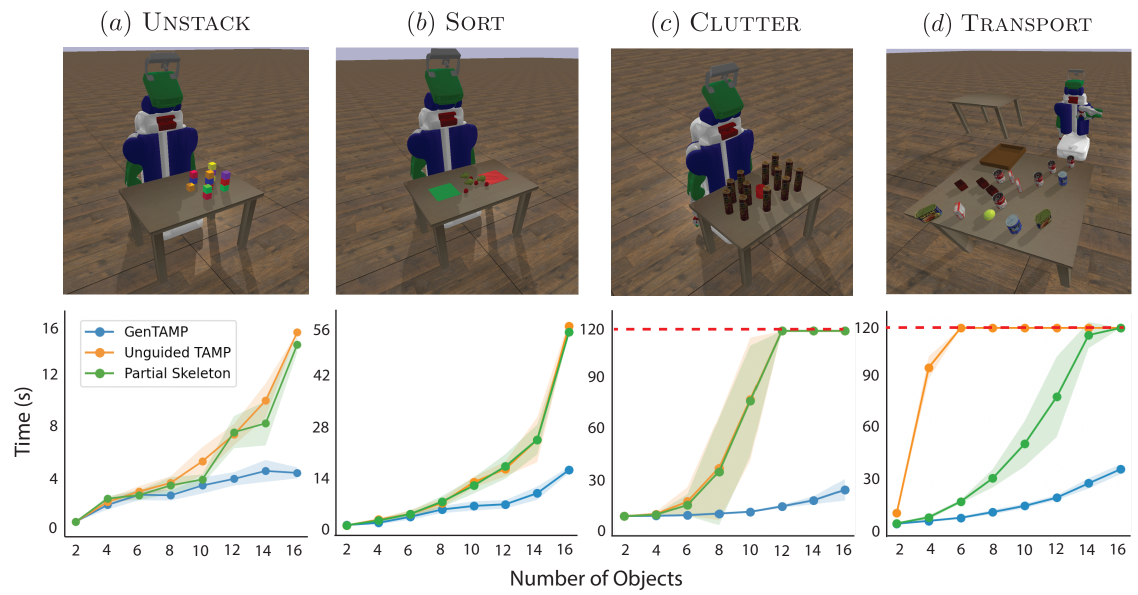

Figure 4-5: The top panels show sample initial states for our continuous TAMP problems. The bottom four panels show planning speed comparisons between our approach, standard TAMP, and partial skeleton. Each trial contains six seeds used to randomize object type, placement, size, and color. The plot error bars show a 95% confidence interval. The red line indicates a timeout.

for each task. The actions in the skeleton lack populated object parameters, reducing the problem to constraint satisfaction. We consider four tasks within a domain containing a PR2 robot with comprehensive kinematic constraints.

In the **Unstack** task, the objective is to remove all stacked blocks and position them on the table's surface. The policy discovered by GenTAMP involves locating a block without any other blocks stacked on top, picking it up, and placing it on the table until no stacked blocks are left.

The goal of the **Sort** task is to arrange all objects on the table into their corresponding colored regions. The policy discovered by GenTAMP consists of placing all green objects in the green region, followed by placing all red objects in the red region. This policy is identified even though none of the example plans sort objects in this specific color-based order.

For the **Clutter** task, the aim is to pick up the target object, which is complicated by obstructions from numerous distractor objects. The policy discovered by GenTAMP is to detect objects that collide when reaching for the target object and move those objects out of the way by placing them elsewhere on the table until the target object can be reached. This task is notoriously difficult for TAMP solvers and has previously required significant engineering, including collision error feedback, to be feasible in large problems [98]. GenTAMP circumvents such engineering by learning features from small examples that guide search towards moving objects blocking trajectories.

In the **Transport** task, the objective is to transfer all objects from the source platform to the destination platform. The agent can utilize a tray to transport multiple objects to the destination region simultaneously. The strategy discovered by GenTAMP is to place objects onto the tray until no more physical space is available, carry the tray to the destination location, and unload the tray until no objects remain at the source location. Since objects vary in size, the generalized TAMP plan considers geometry to determine if more objects can be placed on the tray.

Our experiments show that execution speed from the GenTAMP policy greatly outperforms a state-of-the-art TAMP algorithm. Planning speed increases exponentially with object count, as can be seen in Figure 4-5.

Chapter 5

Long-Horizon Planning with Partial Observability

5.1 Background

One of the constraints of the Task and Motion Planning (TAMP) problem formulation is its reliance on the full observability assumption. This can limit the potential of TAMP-based systems to explore unfamiliar environments, as it requires extensive knowledge about the world state to either be known in advance or gathered through initial observation, a requirement that can prove challenging in scenarios such as navigating unknown environments.

Robust and safe navigation is a necessary capability for mobile-base robots. However, robots with limited sensing abilities, for instance, a single-camera vision system

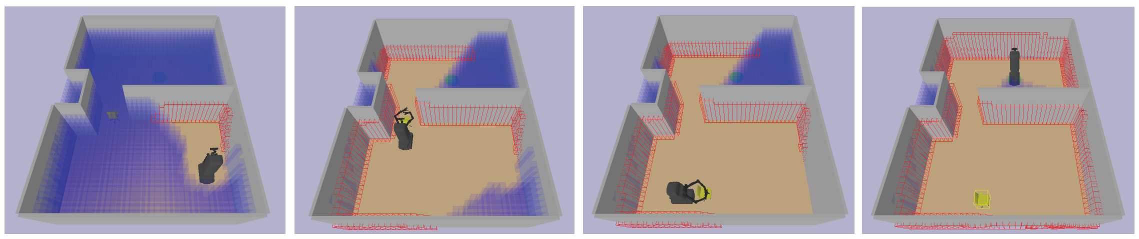

Figure 5-1: A partial simulated execution of a VANAMO task showing the unviewed regions (blue), observed static obstacles (red), and observed movable obstacles (yellow)

and an omnidirectional base, might not always possess a comprehensive view of their workspace. This results in workspace regions that the robot has neither observed nor can access due to safety and reliability considerations. Traditional heuristics that necessitate the robot to visually scan a region before venturing into it might be inadequate when the robot's vision is obstructed by the objects being manipulated or when specific areas of the workspace can only be viewed from certain angles. To overcome this challenge, many papers have proposed algorithms for Visibility-Aware Motion Planning (VAMP), a problem that, despite being NP-hard, has effective solutions under many realistic problem constraints. In scenarios where the task involves moving obstructing objects to reach the goal, additional constraints become relevant, which is addressed under the domain of Navigation Among Movable Obstacles (NAMO).

The VANAMO Problem

Integrating NAMO with VAMP gives rise to a novel problem category, denoted as Visibility-Aware Navigation Among Movable Obstacles (VANAMO). This problem demands reasoning about both object movability and partial observability. In a paper that we recently published, we introduced an algorithm to tackle VANAMO problems, utilizing goal backchaining and visibility reasoning [21]. The details of this algorithm will be expanded upon in this chapter, but an illustration of the algorithm's execution can be seen in figure 5-1.

The VANAMO problem is defined by a tuple $\langle \mathcal{C}, \mathcal{W}, \mathcal{O}, \mathcal{M}, \mathcal{I}, q_g, \mathcal{A}, f, \text{Obs} \rangle$. Here, $\mathcal{C}$ represents the configuration space of the robot, while $\mathcal{W}$ stands for the workspace of the robot, typically a 3D space with defined bounds. $\mathcal{O}$ is a collection of static obstacles, and $\mathcal{M}$ is a set of movable objects. Every $o \in \mathcal{O}$ and $m \in \mathcal{M}$ is depicted by an object shape sharing the same dimensionality as $\mathcal{W}$. $\mathcal{I}$ specifies the state of the world, including the initial robot configuration q_0 and the pose of all static obstacles q_o for each $o \in \mathcal{O}$, and movable obstacles $q_{\Updownarrow}$ for every $m \in \mathcal{M}$. The navigation goal is symbolized by q_g. The robot also possesses an action space $\mathcal{A}$. The function $f : \mathcal{C} \times \mathcal{A} \to C$ represents the dynamics function, mapping a robot configuration q_t and action a_t to a new configuration q_{t+1}. Finally, the observation function Obs : $\mathcal{C} \rightarrow \text{Img}(q_{\mathcal{M}}, q_{\mathcal{O}})$ maps a configuration q to an image projection of the environment, providing partial information of the movable and static objects. In situations where the object shapes in $\mathcal{O}$ and $\mathcal{M}$ are pre-known, the observation function is defined as Obs : $\mathcal{C} \rightarrow \mathcal{P}((i, q_i) \mid i \in \mathcal{M} \cup \mathcal{O})$, with $\mathcal{P}$ being the power set.

In our experimental setup, a holonomic robot acts within an $SE(2)$ configuration space, with the degrees of freedom corresponding to x, y movement and θ rotation. The Obs function is fully characterized by the intrinsics of a forward-facing camera, which maintains a fixed pose relative to the robot base. Despite having a holonomic base, it is crucial to explicitly model θ as it determines the camera direction for directional visibility.

The robot can engage with the environment through various controllers. The `move` controller alters the x, y, and θ dimensions by adding or subtracting to them in fixed increments. The `pick` controller forms a rigid attachment with a movable object, provided that the object is within an ϵ distance of the robot, the bounding box for the movable object is smaller than a specified maximum height and width (i.e., it can be enveloped by the robot's arms), and a part of the object shape is within a certain θ deviation from the robot's orientation (i.e., the robot must be facing the object). The `place` controller eliminates a rigid attachment, if present. The `push` controller operates on movable objects of any size that the camera is within ϵ distance of, as long as some part of the object is within a certain θ deviation from the robot's orientation.

5.2 Method

In the following section, we present an algorithm designed to tackle VANAMO problems, termed Look and Manipulate Backchaining (LAMB). This algorithm utilizes a hierarchical search approach with two levels. The primary or lower level involves an A* algorithm which functions by taking an initial configuration, goal configuration, an attachment (should it exist) and its relative pose, along with a set of obstacles

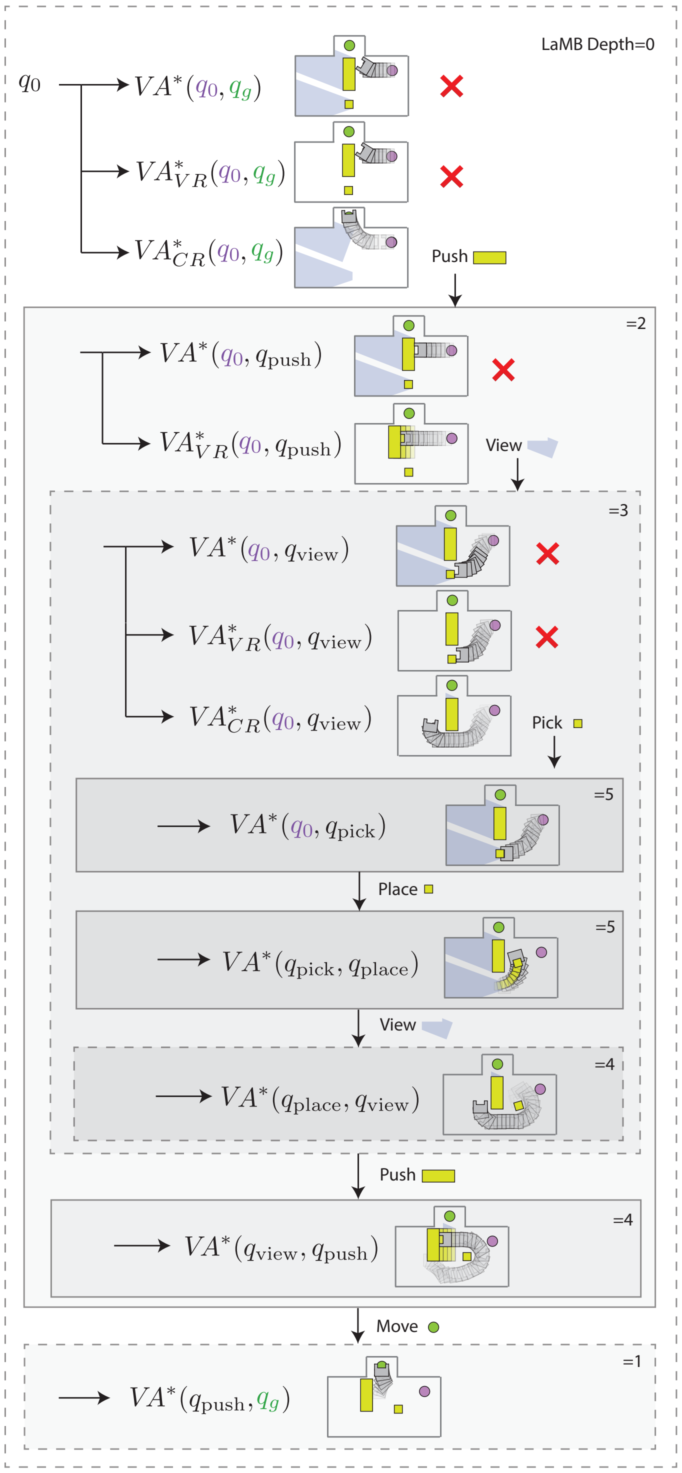
LaMB Depth=0
q_0
$VA^*(q_0, q_g)$
$VA^*_{VR}(q_0, q_g)$
$VA^*_{CR}(q_0, q_g)$
Push
=2
$VA^*(q_0, q_{\text{push}})$
$VA^*_{VR}(q_0, q_{\text{push}})$
View
=3
$VA^*(q_0, q_{\text{view}})$
$VA^*_{VR}(q_0, q_{\text{view}})$
$VA^*_{CR}(q_0, q_{\text{view}})$
Pick
=5
$VA^*(q_0, q_{\text{pick}})$
Place
=5
$VA^*(q_{\text{pick}}, q_{\text{place}})$
View
=4
$VA^*(q_{\text{place}}, q_{\text{view}})$
Push
=4
$VA^*(q_{\text{view}}, q_{\text{push}})$
Move
=1
$VA^*(q_{\text{push}}, q_g)$

Figure 5-2: An example trace of the LAMB algorithm on the **obstructed visibility** task. We denote each nested recursive call with a darker shade of grey. The start location is marked with a purple circle, and the goal location is marked with a green circle. VA$^*_{VR}$ denotes vision-relaxed motion planning, and VA$^*_{CR}$ denotes movable object relaxed motion planning. For clarity, we collapse successive move & manipulate calls and denote them with solid lines.

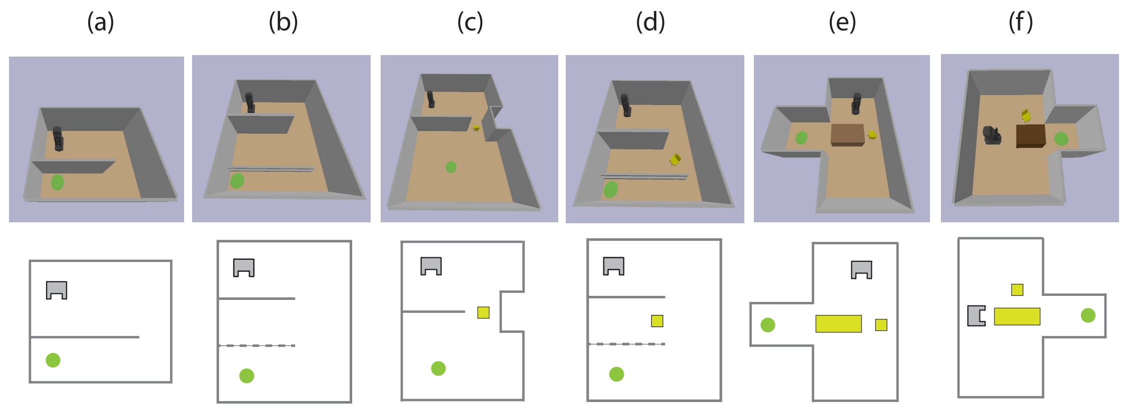

Figure 5-3: **Problem Instance Visualization.** A picture of problem instances from each task category (top row) with illustrated depictions of the envioronments for visual clarity (bottom row). The green dot indicates the goal position, the yellow objects in the botom row indicate movable objects. The large box is pushable, but not directly graspable.

that need to be avoided. The goal often can't be reached directly via motion planning, so the secondary or higher level search identifies a sequence of navigation and manipulation actions to eliminate the constraints that hinder goal reachability.

Two types of constraints impede navigation. *Visibility constraints* guarantee that the robot never traverses through or moves obstacles into a region that has not been observed. *Collision constraints* ensure the robot never navigates or moves obstacles through areas filled with obstacles. The LAMB algorithm conducts a high-level search through look actions (considered as `move` actions for the purposes of looking) that discard visibility constraints, and manipulation actions that displace objects, thereby modifying collision constraints. Interdependencies between these configurations occasionally result in one visibility or collision constraint inhibiting the removal of another constraint. To manage these intricate interactions, the higher-level search decomposes goals into subgoals with an aim to eliminate each constraint individually.

The LAMB planner functions within the execution context specified in algorithm 6. Visibility, static occupancy, and movable occupancy grids are initialized as empty before any observations.

The observations are represented as segmented point clouds, with segments being labeled as per their object index. This segmentation offers insights about whether a specific object is movable. Realistically, this could be inferred from the object's material or shape properties. However, in our experiments and simulations, the objects are identified using ground truth per-pixel object identities from the simulation. The visibility grid for a given configuration, symbolized as $\text{Vis}(q)$, is calculated by casting rays from the camera to the point delineated by the captured depth image, marking all traversed voxels as viewed. The LAMB algorithm is then initiated with the updated grids along with the navigation goal and the initial state. The first action from the plan returned by LAMB is executed in the environment, and the fresh observation is employed to update the grids. This cycle continues until the goal is achieved or the process times out.

Within LAMB , we account for three scenarios: direct planning, visibility-relaxed planning, and collision-relaxed planning, as delineated in Algorithm 6. In each scenario, VA* is used to assess if the goal is reachable with various degrees of relaxed constraints. VA* is a modified variant of A^* that efficiently manages path-dependent visibility. Specifically, once a path to a certain state is identified, we disregard any shorter paths to that state, and we link each state with the visibility grid derived from the path that first arrives at that state. This may not always lead to optimal visibility-based paths but serves as an effective approximation [45]. As the planner can't predict the observations the robot will make as it traverses a path, we cannot ascertain the visibility of an imagined configuration. Consequently, we employ optimistic visibility. In other words, within VA*, we assume that voxels not known to contain an obstacle are unoccupied space. If a direct path with VA* is feasible, we return that path as the plan. In the event that a direct path isn't viable, LAMB initially attempts to ease visibility constraints. This is achieved by planning with VA* and an empty visibility grid. If a plan is discovered under these relaxed constraints, we determine the region that needs to be viewed by intersecting the swept volume of the path, denoted as SWEPT(path), with the existing visibility and generate a subgoal for viewing that area. Often, the necessary region cannot be seen from a single perspective. As a result, we employ a unique heuristic that drives progress towards partially viewing the required region. By calculating a scalar field, F, in our workspace, which represents the shortest distance from any point in the workspace to the needed region, we define our new heuristic as $\mathrm{H}F(q) = \min x \in \mathrm{Vis}(q)F(x)$. The new subgoal is obtained by running our VA* algorithm in an obstacle-relaxed setting. Every subgoal demands a separate plan, which we acquire using a recursive call to LaMB. This call must be recursive because additional constraints, like obstructing movable obstacles, might hinder the reachability of the necessary viewing positions. In our experiments, the robot's configuration is set to always see its base. This configuration lowers the number of visibility subgoals needed for task completion but isn't strictly essential for our algorithm to function.

If a visibility-relaxed plan is unattainable, LaMB calculates a collision-relaxed plan that eliminates collision constraints enforced by *movable* obstacles. This is done by setting the movable occupancy grid to empty and planning a path to the goal using VA*. If a path is identified, the first movable object collision is detected using the same Swept subprocedure employed for computing the visibility subgoal. In this work, we focus on moving only the first obstacle the robot encounters along a path to the goal. This method assumes we are addressing LP_1 NAMO problems [69]. Although this algorithm can be extended to consider multiple obstacles, doing so would significantly increase computational cost and isn't required for our environments. Nevertheless, we do examine multiple ways of interacting with the object. For each object, we consider `push` and `pick`/`place` operations (depending on the object's size) from various grasp locations. Pushing is a more restricted operation but may be necessary if the object is too wide for the robot to encircle with its arms. For each manipulation action considered, there is a q_{pre} and q_{post} robot configuration. With these intermediate configurations, we can plan a path to manipulate the object and subsequently reach the goal through recursive calls to LaMB. Additionally, we need to compute the updated occupancy grid and swept volume. The updated occupancy grid is crucial for planning after manipulation, while the swept volume is vital for planning before manipulation. The swept volume introduces a constraint to the planner that prevents moving other obstacles into the swept path of the manipulated object prior to its manipulation. For more information on this approach, see [101].

If no plan can be identified under these relaxations, the planner is terminated and a failure result is returned. Figure 5-2 illustrates an example trace of this algorithm on one of the more intricate tasks involving multiple recursive calls with both visibility and collision relaxation.

Algorithm 4 $\textsc{EvaluatePlanner}(\mathcal{O}, \mathcal{M}, \mathcal{I}, q_g, \mathcal{A}, f, \text{Obs})$

```
1: $q_t \leftarrow q_0$
2: $\text{GridV} \leftarrow \emptyset, \text{Grid}\mathcal{O} \leftarrow \emptyset, \text{Grid}\mathcal{M} \leftarrow \emptyset$
3: while $\neg(q_t = q_g)$ do
4:     $o \leftarrow \text{Obs}(q_t)$
5:     $\text{GridV} \leftarrow \textsc{UpdateVis}(\emptyset, o)$
6:     $\text{Grid}\mathcal{O}, \text{Grid}\mathcal{M} \leftarrow \textsc{UpdateObs}(\text{Grid}\mathcal{O}, \text{Grid}\mathcal{M}, o)$
7:     $\text{plan} \leftarrow \text{LAMB}(q_0, q_g, f, \text{Grid}\mathcal{O}, \text{Grid}\mathcal{M}, \text{GridV})$
8:     $q_t \leftarrow f(q_t, \text{plan}[0])$
9:     $\text{trace} \leftarrow \text{trace} \oplus (plan[0], q_t)$
10: return trace
```

Algorithm 5 $\text{VA}^*(q_0, G, \text{G}\mathcal{O}, \text{G}\mathcal{V}, \text{H}(q) = ||q - q_g||_2)$

```
1: $Q \leftarrow [[q_0]], \text{visited} \leftarrow \emptyset$
2: while $|Q| \neq 0$ do
3:     $\text{path} \leftarrow Q.\text{pop}(0), q_{\text{last}} \leftarrow Q[0][0]$
4:     if $q_{\text{last}} \in G$ then
5:         return path
6:     if $q_{\text{last}} \in \text{visited}$ then
7:         continue
8:     $\text{V} \leftarrow \text{PathVision}(\text{path}, \text{G}\mathcal{O})$
9:     $V' \leftarrow G\mathcal{V} \cup V$
10:    $N_q \leftarrow \textsc{Neighbors}(q_{\text{last}})$
11:    $Q \leftarrow \{\text{path} \oplus [q'] \mid q' \in \text{N}_q, \textsc{Swept}([q']) \cap V')^c = \emptyset\}$
12:    $Q \leftarrow \textsc{Sorted}(Q, \text{key} = \text{Cost}(\text{path}) + \text{H}(q_{\text{last}}))$
13:    $\text{visited} \leftarrow \text{visited} \cup \{q_{\text{last}}\}$
```

5.3 Experiments

In order to showcase the relevance of visibility reasoning in NAMO and assess our algorithm, we have created a set of five task categories. Each of these categories presents unique challenges. The initial state for each category is displayed in Figure 5-3. The objective of each task category is to navigate to a particular area highlighted in

Algorithm 6 LaMB($q_0, q_g, G\mathcal{O}, G\mathcal{M}, G\mathcal{V}$)

```
plan ← $\text{VA}^*(q_0, \{q_g\}, G\mathcal{O} \cup G\mathcal{M}, \emptyset, \text{GV})$    ▷ Direct
if plan then
    return plan
GS ← $G\mathcal{O} \cup G\mathcal{M}$
plan ← $\text{VA}^*(q_0, \{q_g\}, G\mathcal{O} \cup G\mathcal{M}, \emptyset)$    ▷ Visibility-Relaxed
if plan then
    $\text{V}_{sg}$ ← Swept(plan) $\cap\, G\mathcal{V}^c$
    $q \leftarrow q_0$
    for $q' \in \text{VA}^*(q, V_{sg}, G\mathcal{O}, \text{H} = H_F)$ do
        plan ← plan $\oplus$ LaMB($q_0, q', G\mathcal{O}, G\mathcal{M}, \text{GV}$)
        $q \leftarrow q'$
    plan ← plan $\oplus$ LaMB($q, q_g, G\mathcal{O}, G\mathcal{M}, \text{GV}$)
    if None $\notin$ plan then
        return plan
plan ← $\text{VA}^*(q_0, \{q_g\}, \emptyset, \text{GV})$    ▷ Collision-Relaxed
if plan then
    Obj ← FirstCollision(plan, $G\mathcal{O}$)
    GS ← $G\mathcal{O} \cup G\mathcal{M} \setminus$ {Object}
    for $q_{pre}, q_{post} \in$ SampleManip(Obj, GS) do
        mid ← LaMB($q_{pre}, q_{post}, G\mathcal{O}, G\mathcal{M}\backslash\text{Obj}, \text{GV}$)
        $G\mathcal{O}'$ ← $G\mathcal{O} \cup$ Swept(plan, Obj)
        pre ← LaMB($q_0, q_{pre}, G\mathcal{O}', G\mathcal{M}\backslash\text{Obj}, \text{GV}$)
        $G\mathcal{O}''$ ← UpdatePose(Obj, $G\mathcal{O}$)
        post ← LaMB($q_{post}, q_g, G\mathcal{O}'', G\mathcal{M}\backslash\text{Obj}, \text{GV}$)
        if None $\notin$ pre $\oplus$ mid $\oplus$ post then
            return pre $\oplus$ mid $\oplus$ post
return None
```

green. Within each category, we experiment with random placement of objects, robot positions, and goal region locations, adhering to the constraints of the respective task category. Here we detail each task category and the baselines incorporated in our evaluation.

5.3.1 Task Categories

The **Simple Navigation (T1)** task category is the simplest, where the robot can reach the navigation goal without needing to move any obstacles (Figure 5-3a). **Visibility (T2)** tasks take inspiration from issues found in the visibility-aware motion planning literature [45, 53, 26, 5, 71]. In these tasks, direct navigation to the goal is not possible due to visibility constraints. Figure 5-3b illustrates a situation where the robot can only traverse the hallway sideways, hence it must observe the hallway from outside before navigating through it to avoid colliding with unseen areas. The **Movable Obstacles (T3)** tasks represent standard NAMO problems featuring movable obstacles that are entirely visible from the initial state. These tasks generally do not require additional visibility reasoning (Figure 5-3c). The **Obstructed Visibility (T4)** tasks have visibility constraints similar to the visibility tasks, but to obtain necessary observations, one or more obstacles must be moved (Figure 5-3e). **Occluding Obstacles (T5)** tasks include movable objects that partially or completely block the robot's view during interaction. Resolving these tasks frequently requires viewing certain regions before interacting with an object, as shown in Figure 5-3d. Finally, **Obstructed Affordances (T6)** tasks necessitate the manipulation of obstacles from configurations that cannot be accessed without manipulating other obstacles. In the example given in Figure 5-3f, the box object needs to be pushed but it cannot be done directly as it would block the goal. The box also cannot be immediately pushed from the bottom due to visibility constraints at the top. The robot has to first move the obstructing chair and then push the box from either the bottom or top.

	T1	T2	T3	T4	T5	T6
VA*	**5/5**	0/5	0/5	0/5	0/5	0/5
NAMO	**5/5**	0/5	4/5	0/5	0/5	0/5
FO-NAMO	**5/5**	0/5	**5/5**	0/5	0/5	0/5
VAMP	**5/5**	**5/5**	0/5	0/5	0/5	0/5
LaMB	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**

Figure 5-4: Experimental results

5.3.2 Baselines

We compared LaMB with four search baselines that include visibility constraints. The **VA-Star** baseline executes an VA* search in the discretized configuration space using a distance-to-goal heuristic. An extra constraint was imposed on the VA* search, limiting actions to those that do not pass through unobserved regions. The A* baseline was only successful on the simple navigation task where the heuristic was a useful metric. When direct navigation to the goal wasn't feasible, VA* would default to a comprehensive search until a timeout occurred.

The **Fully Observable NAMO** baseline offers a solution for fully observable NAMO problems [81]. Initially, this baseline identifies a relaxed path to the goal by navigating through movable obstacles. It then contemplates transfer paths for each obstacle in reverse collision order, starting from the goal. For each movable object the planner engages with, it introduces artificial collision constraints for motion on the subsequent obstacle it attempts to maneuver. The search process follows a depth-first approach where motion constraints deemed unfeasible mark the end of search nodes. The backward planning approach from the goal makes it impossible to enforce visibility constraints, leading this baseline to assume all unseen space is unoccupied. Our findings indicate that this baseline is effective when obstacles block all paths to the goal, but fails when visibility constraints limit obstacle movement.

The **Constrained-NAMO** baseline draws inspiration from related research in visual NAMO [108]. This algorithm iterates through all identifiable visible objects, assesses if relocating an object would result in a more direct path to the goal, and if so, relocates the object. Much like the VA* baseline, visibility constraints are imposed on the low-level configuration-space search. Our data show that this baseline fails when moving obstacles directly is unfeasible due to visibility or obstruction.

The **VAMP** baseline represents a cutting-edge algorithm for visibility-aware motion planning [45]. Similar to FO-NAMO and LAMB , VAMP conducts back chaining from the goal by establishing intermediate subgoals based on failure from relaxed goal planning. Rather than setting subgoals that involve movable obstacle manipulation, VAMP initially tries to plan straight to the goal while disregarding vision constraints. It then identifies the workspace regions that were navigated through but not observed and sets viewing these regions as a subgoal. Since VAMP does not consider setting object manipulation subgoals, it only succeeds on simple navigation and visibility tasks.

Our findings demonstrate that LAMB is the only algorithm capable of solving the last three tasks, which each require some degree of reasoning about the relationship between visibility and manipulation.

5.3.3 Results

Each task was executed with five distinct seeds that define a unique initial position for the robot and obstacles. The total success rate out of these five runs is reported in Table 5-4. As anticipated, the VA* baseline was only successful in solving simple navigation tasks. VA*, in contrast to other motion planning algorithms, is not probabilistically complete due to its reliance on visibility-based path dependence. This probabilistic incompleteness results in certain failure when visibility constraints are not satisfied via the shortest path to a configuration. Like other motion planning algorithms, VA* slows down significantly with the increasing dimensions of the configuration space introduced by additional movable obstacles, leading to a timeout on the NAMO problems. The Fully Observable and Constrained NAMO baselines mostly succeeded in the simple navigation and movable obstacles tasks. However, the Fully Observable baseline occasionally fails on the NAMO task because it attempts to position movable objects in unobserved regions or navigate through unseen regions while carrying an object. These two baselines failed in all other tasks as they did not regard visibility as a potential subgoal. The VAMP baseline succeeded in all tasks where visibility was the only constraint (simple navigation and visibility constrained) but failed when placed in an environment where obstacle movement was necessary. LaMB demonstrated success on all seeds for each task.

Chapter 6

Conclusion

This thesis introduces a novel approach to long-horizon robotic planning using visual input. This approach overcomes the constraints of existing Task and Motion Planning (TAMP) methodologies by facilitating the automatic construction of problems without prior knowledge of object models, their properties, or their affordances. The system has demonstrated its effectiveness by generalizing to unique object sets and scene configurations in several real-world robotic contexts, underscoring its potential to resolve TAMP issues and address the fundamental challenge of proficient world modeling and planning from perception.

Furthermore, this work has addressed numerous limitations inherent in this framework. Specifically, it enhances the framework's capacity to manage large sets of objects, encompassing scenarios with a handful of distinguishable relevant objects as well as situations with numerous indistinguishable but relevant objects. We also introduced an algorithm that actively seeks crucial world information and refines the constructed abstraction based on prior experience, thereby enabling efficient future planning.

These contributions chart a path for future research aimed at developing advanced, intelligent robotic systems capable of autonomous operation in complex environments.

Bibliography

[1] David Abel, D Ellis Hershkowitz, and Michael L Littman. Near optimal behavior via approximate state abstraction. *arXiv preprint arXiv:1701.04113*, 2017.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] Rabab Benotsmane, László Dudás, and György Kovács. Trajectory optimization of industrial robot arms using a newly elaborated "whip-lashing" method. *Applied Sciences*, 10(23), 2020.

[4] Jonathan Binney, Andreas Krause, and Gaurav S. Sukhatme. Informative path planning for an autonomous underwater vehicle. In *2010 IEEE International Conference on Robotics and Automation*, pages 4791–4796, 2010.

[5] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon "next-best-view" planner for 3d exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, 2016.

[6] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):281–300, 1997.

[7] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Larry Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. 04 2016.

[8] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. *CoRR*, abs/1811.07231, 2018.

[9] Blai Bonet, Raquel Fuentetaja, Yolanda E-Martín, and Daniel Borrajo. Guarantees for sound abstractions for generalized planning (extended paper). *CoRR*, abs/1905.12071, 2019.

[10] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. *CoRR*, abs/1801.10055, 2018.

[11] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[12] Daniel Bryce and Olivier Buffet. International planning competition uncertainty part: Benchmarks and results. In *In Proceedings of IPC*, 2008.

[13] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. *CoRR*, abs/1606.02378, 2016.

[14] 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 *ICAR*, 2015.

[15] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Kenneth Goldberg, Pieter Abbeel, and Nathan Michael. Information-theoretic planning with trajectory optimization for dense 3d mapping. 07 2015.

[16] Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. *CoRR*, abs/2105.14074, 2021.

[17] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer, Berlin, 5 edition, 2003.

[18] Augusto B Corrêa, Florian Pommerening, Malte Helmert, and Guillem Frances. Lifted successor generation using query optimization techniques. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 80–89, 2020.

[19] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2016.

[20] Aidan Curtis, Xiaolin Fang, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Caelan Reed Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. *CoRR*, abs/2108.04145, 2021.

[21] Aidan Curtis, Jose Muguira-Iturralde, Yilun Du, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Visibility-aware navigation among movable obstacles, 2022.

[22] Aidan Curtis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Discovering state and action abstractions for generalized task and motion planning. *CoRR*, abs/2109.11082, 2021.

[23] Richard Dearden and Craig Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.

[24] Rosen Diankov. *Automated construction of robotic manipulation programs*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2010.

[25] Thomas G Dietterich. State abstraction in maxq hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 994–1000, 2000.

[26] Christian Dornhege and Alexander Kleiner. A frontier-void-based approach for autonomous exploration in 3d. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 351–356, 2011.

[27] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

[28] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4), 1983.

[29] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *ArXiv*, abs/2006.08381, 2020.

[30] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[31] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *CVPR*, 2020.

[32] Peiyuan Fang, Zhuoping Yu, Lu Xiong, Zhiqiang Fu, Zhuoren Li, and Dequan Zeng. A maximum entropy inverse reinforcement learning algorithm for automatic parking. In *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, pages 1–6, 2021.

[33] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G. Atkeson. Optimization based full body control for the atlas robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 120–127, 2014.

[34] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, 2017.

[35] Peter R. Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian S. Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. *ArXiv*, abs/2109.00137, 2021.

[36] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.

[37] Anurag Ganguli, Jorge Cortés, and Francesco Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *IEEE Transactions on Robotics*, 25:340–352, 2006.

[38] C. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Perez. Integrated task and motion planning. *Annual Review: Control, Robotics, Autonomous Systems*, 4, 2021.

[39] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers. In *ICAPS*, 2020.

[40] 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 *ICRA*, 2020.

[41] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Stripstream: Integrating symbolic planners and blackbox samplers. *CoRR*, abs/1802.08705, 2018.

[42] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[43] Daniel Gnad, Alvaro Torralba, Martín Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7602–7609, 2019.

[44] Gustavo Goretkin, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Look before you sweep: Visibility-aware motion planning. *CoRR*, abs/1901.06109, 2019.

[45] Gustavo Goretkin, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Look before you sweep: Visibility-aware motion planning, 2019.

[46] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2018.

[47] M. Gualtieri, A. T. Pas, K. Saenko, and R. Platt. High precision grasp pose detection in dense clutter. *IROS*, 2016.

[48] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel. Modular task and motion planning in belief space. *IROS*, 2015.

[49] Sebastien B. Hausmann, Alessandro Marin Vargas, Alexander Mathis, and Mackenzie W. Mathis. Measuring and modeling the motor system with machine learning. *Current Opinion in Neurobiology*, 70:11–23, 2021. Computational Neuroscience.

[50] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

[51] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[52] Malte Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246, jul 2006.

[53] Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1071–1078, 2015.

[54] Natalia Hernandez-Gardiol. *Relational envelope-based planning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008.

[55] Sabir Hossain and Deok-jin Lee. Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices. *Sensors*, 19(15), 2019.

[56] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7610–7618, Jul. 2019.

[57] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39:407–428, 10 2015.

[58] L. Kaelbling and T. Lozano-Perez. Integrated task and motion planning in belief space. *IJRR*, 32, 2013.

[59] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[60] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE, 2011.

[61] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[62] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. *CoRR*, abs/1709.10489, 2017.

[63] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018.

[64] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[65] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439, 2020.

[66] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1194–1201. AAAI Press, 1996.

[67] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[68] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[69] James J. Kuffner and Mike Stilman. Navigation among movable obstacles. 2007.

[70] James J Kuffner Jr. and Steven M LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[71] Mikko Lauri and Risto Ritala. Planning for robotic exploration based on forward simulation. *Robotics and Autonomous Systems*, 83, 02 2015.

[72] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, pages 293–308, 2001.

[73] Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *Ellis Horwood Series in Artificial Intelligence*, pages 146–160. Springer, 1994.

[74] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.

[75] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[76] Minghua Liu, Lu Sheng, Sheng Yang, Jing Shao, and Shi-Min Hu. Morphing and sampling network for dense point cloud completion. In *AAAI*, volume 34, 2020.

[77] Lingni Ma, Jörg Stückler, Christian Kerl, and Daniel Cremers. Multi-view deep learning for consistent semantic mapping with rgb-d cameras. pages 598–605, 09 2017.

[78] Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Online planner selection with graph neural networks and adaptive scheduling. In *AAAI*, pages 5077–5084, 2020.

[79] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. pages 3932–3939, 09 2017.

[80] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 438–445, Cham, 2014. Springer International Publishing.

[81] Shokraneh K. Moghaddam and E. Masehian. Planning robot navigation among movable obstacles (namo) through a recursive approach. *Journal of Intelligent & Robotic Systems*, 83:603–634, 2016.

[82] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. *CoRR*, abs/1905.10520, 2019.

[83] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[84] Christian Muise, Sheila A. McIlraith, and J. Beck. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*, 2012.

[85] Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[86] A. Nguyen, D. Kanoulas, D. G Caldwell, and N. G Tsagarakis. Object-based affordances detection with convolutional neural networks and dense conditional random fields. In *IROS*, 2017.

[87] Edwin PD Pednault. ADL: Exploring the middle ground between strips and the situation calculus. *Kr*, 89:324–332, 1989.

[88] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[89] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[90] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *PAMI*, 39(6), 2016.

[91] Bernardus Ridder. *Lifted heuristics: towards more scalable planning systems.* PhD thesis, King's College London (University of London), 2014.

[92] Or Rivlin, Tamir Hazan, and Erez Karpas. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, 2020.

[93] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[94] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2020.

[95] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics.* Springer-Verlag, Berlin, Heidelberg, 2007.

[96] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*, 2020.

[97] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. *CoRR*, abs/2009.05613, 2020.

[98] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, 2014.

[99] Siddharth Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175:615–647, 2011.

[100] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 1010–1016, San Francisco, California, 2011.

[101] Mike Stilman and James J. Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27:1295 – 1307, 2006.

[102] Markus Suchi, Timothy Patten, David Fischinger, and Markus Vincze. Easy-Label: A semi-automatic pixel-wise object annotation tool for creating robotic RGB-D datasets. In *ICRA*, 2019.

[103] Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.

[104] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

[105] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, 2015.

[106] Marc Toussaint, Kelsey R. Allen, K. Smith, and J. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.

[107] L. Wong, L. Kaelbling, and T. Lozano-Perez. Manipulation-based active search for occluded objects. *ICRA*, 2013.

[108] Hai-Ning Wu, Martin Levihn, and Mike Stilman. Navigation among movable obstacles in unknown environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1433–1438, 2010.

[109] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[110] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *RSS*, 2018.

[111] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. In *arXiv:2007.08073*, 2020.

[112] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997.

[113] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. 2018.