Title

Scene Abstraction for Generalizable Long-horizon Robot Planning

Author

Han, Muzhi

Publication Date

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Scene Abstraction for Generalizable Long-horizon Robot Planning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Mechanical Engineering

by

Muzhi Han

2024

ABSTRACT OF THE DISSERTATION

Scene Abstraction for Generalizable Long-horizon Robot Planning

by

Muzhi Han
Doctor of Philosophy in Mechanical Engineering
University of California, Los Angeles, 2024
Professor Song-Chun Zhu, Co-Chair
Professor Dennis Hong, Co-Chair

Humans excel at abstracting raw information into meaningful high-level representations, which builds the foundation for understanding complex situations and making sophisticated decisions in novel scenarios. In contrast, robots often struggle to solve complex tasks and generalize to unseen situations due to their limited abstraction capabilities. This dissertation presents a novel scene abstraction perspective and a holistic framework for robots to perform long-horizon tasks in unseen real-world scenarios by: (i) perceiving scenes as abstract states, (ii) acquiring world models that predict action potentials and consequences on abstract states, and (iii) planning to reach novel goals within the abstract state space using these world models. We advocate for a scene graph-based representation that abstracts objects and their relations as symbols, allowing for strong compositional generalization to novel objects and goals in planning. The dissertation is structured in three parts, with focus on perception, planning, and learning, respectively. In the first part, we introduce a manually-defined contact graph representation that preserves the kinematic state of the environment for task and motion planning. We develop a scene reconstruction system that recovers this representation from RGB-D streams, enabling the creation of functionally-equivalent digital twins

for simulating robot interaction. In the second part, we demonstrate closed-loop reasoning and planning using contact graphs and other feedback forms, leveraging the internal world knowledge of language models. We show that a Vision Language Model (VLM) can enable closed-loop mobile manipulation in the real world with feedback from the contact graph and images from the robot's wrist camera. We also show that an Large Language Model (LLM) can propose Task and Motion Planning (TAMP) solutions and make corrections by reasoning motion planner feedback. The third part focuses on learning task-relevant symbolic abstractions and world models that generalize over novel object configurations. We present an interactive framework that learns PDDL-style symbolic predicates and operators from interaction data and language feedback. Additionally, we propose a probabilistic framework that learns object symbols and a stochastic grammar capturing state transitions in the context of object cutting. We demonstrate that these learned symbolic representations and world models can be utilized to solve complex tasks with novel objects and unseen goals through planning. By placing abstraction at its core, this dissertation seeks to unify perception, planning, and learning to build more capable and generalizable embodied intelligence.

The dissertation of Muzhi Han is approved.

Veronica Santos

Ying Nian Wu

Tsu-Chin Tsao

Dennis Hong, Committee Co-Chair

Song-Chun Zhu, Committee Co-Chair

University of California, Los Angeles

2024

To my family,

whose love and support made this journey possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express my profound gratitude to my advisor, Prof. Song-Chun Zhu, for his invaluable guidance and inspiration throughout my Ph.D. journey. He introduced me to the fascinating world of artificial intelligence and computer vision, encouraging me to aim high and approach problems from a fundamental perspective. Working with Prof. Zhu shapes my career goal - to build capable and general robot intelligence for the welfare of human beings.

I would like to extend my sincere appreciation to all my committee members: Prof. Veronica Santoa, Prof. Tsu-Chin Tsao, Prof. Dennis Hong, and Prof. Ying Nian Wu. Their time, service, and kind advice throughout this process have been invaluable. I am especially grateful to Prof. Hong, my committee co-chair, for his tremendous support in administrative matters and for inspiring me to maintain an open mind and cultivate a passion for my work. My heartfelt thanks also go to Prof. Wu, who is always willing to chat with me about research and career questions. His encouragement and guidance navigate me through the hard time when I was working alone and lost. He is the kindest person I ever knew and I admire him for his serious attitude towards research.

During my Ph.D. career, I have been fortunate enough to be mentored by the best researchers and mentors, Dr. Hangxin Liu, Prof. Yixin Zhu, and Prof. Yuke Zhu. They have shown me how to conduct impactful research and taught me the art of writing high-quality scientific papers and delivering effective presentations. I appreciate their great patience and continued help on me. Under their mentorship, I have grown not only into a mature researcher and qualified Ph.D., but also gained valuable life lessons that extend far beyond research.

My Ph.D. research would not be possible without the talented and helpful people I worked with, Dr. Zeyu Zhang, Dr. Ziyuan Jiao, Dr. Baoxiong Jia, Dr. Siyuan Huang, Dr. Xu Xie, Yifeng Zhu, Shu Wang, Dr. Tengyu Liu, Peiyuan Zhi, Zhiyuan Zhang, Puhao Li, Sixu Yan, and Lexing Zhang. I will always miss the days we work closely, not just as collaborators but also as close friends. I would also thank the best labmates and friends I could imagine, Weiqi Wang, Dr. Xiaojian Ma, Dr. Ran Gong, Dehong Xu, Dr. Sirui Xie, Deqian Kong, Dr. Pan Lu, Qian Long, Dr. Siyuan Qi, Dr.

Qing Li, Dr. Feng Gao, Dr. Yuxing Qiu, Shuwen Qiu, Dr. Ruiqi Gao, Dr. Yaxuan Zhu, Dr. Chi Zhang, Dr. Lifeng Fan, Dr. Mark Edmonds, Dr. Xiaofeng Gao, Dr. Luyao Yuan, and Dr. Zilong Zheng. I really enjoy the time we spent together and I hope to see you again in the future.

I want to acknowledge the UCLA Life Sciences Core TAship program for providing continuing financial support that makes this dissertation possible. It also offers me the opportunity to gain invaluable teaching experience and improve my communication skills. I was supported by TAship for three full years, starting as a teaching assistant and got promoted into a teaching associate and finally a teaching fellow. Teaching and learning with my students was a routine of my Ph.D. life.

Finally, I dedicate this dissertation to my parents and my wife for their unwavering support. My parents spared no efforts to fund me in the first year of my Ph.D., when there was no other funding option available. They have been providing me the best emotional support they could from 7000 miles away. My wife is my best friend who has been accompanying me the past 6 years. The magic girl is everywhere: she helps from coursework to career development, manages our small family effectively and efficiently, and turns my depression into happiness. She navigates me through my hardest moments, and I can't imagine this journey without her. I'm fortunate to marry her in 2023.

VITA

2019	B.E. in Elite Program of Mechanical Engineering, Tsinghua University.
2019–2024	Graduate Research Assistant, UCLA.
2021-2023	Graduate Teaching Assistant / Associate / Fellow, UCLA.
2021	Ph.D. candidate, Mechanical Engineering, UCLA.
2021	Applied Scientist Intern, Amazon Lab126.
2022	Research Scientist Intern, Meta Reality Labs Research.
2023	Research Scientist Intern, Beijing Institute for General Artificial Intelligence (BIGAI).
2024	Applied Scientist Intern, Amazon Robotics.

PUBLICATIONS

(* indicates equal contribution)

M. Han, Y. Zhu, S.-C. Zhu, Y. Wu, Y. Zhu. InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning, *Robotics: Science and Systems (**RSS**)*, 2024

S. Wang*, **M. Han***, Z. Jiao*, Z. Zhang, Y. Wu, S.-C. Zhu, H. Liu. LLM^3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning, *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, 2024

P. Li*, T. Liu, Y. Li, **M. Han***, H. Geng, S. Wang, Y. Zhu, S.-C. Zhu, S. Huang. Ag2Manip: Learning Novel Manipulation Skills with Agent-Agnostic Visual and Action Representations, *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, 2024

Z. Zhang*, **M. Han***, B. Jia, Z. Jiao, Y. Zhu, S.-C. Zhu, and H. Liu. Learning a Causal Transition Model for Object Cutting, *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, 2023

Z. Zhang*, L. Zhang*, Z. Wang, Z. Jiao, **M. Han**, Y. Zhu, S.-C. Zhu, and H, Liu. Part-Level Scene Reconstruction Affords Robot Interaction, *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, 2023

M. Han*, Z. Zhang*, Z. Jiao, X. Xie, Y. Zhu, S.-C. Zhu, and H. Liu. Scene Reconstruction with Functional Objects for Robot Autonomy, *International Journal of Computer Vision (**IJCV**)*, 2022

M. Han*, Z. Zhang*, Z. Jiao, X. Xie, Y. Zhu, S.-C. Zhu, and H. Liu. Reconstructing Interactive 3D Scenes by Panoptic Mapping and CAD Model Alignments, *IEEE International Conference on Robotics and Automation (**ICRA**)*, 2021

CHAPTER 1

Introduction

Humans have long envisioned robotic assistants capable of handling complex household tasks and adapting to diverse scenarios [Nil84, Gat07]. Consider a household robot tasked with preparing a meal in the kitchen. It must devise an intricate sequence of actions to locate and retrieve ingredients, manipulate various utensils, and adapt these steps based on potentially varied recipes, object configurations, and cooking goals. While such tasks are manageable for humans, they pose significant challenges to robots: (i) The **long-horizon** [ZTB21, CFK22] nature of the task requires the robot to maintain context and execute a lengthy sequence of interdependent actions, adapting to intermediate outcomes throughout the process. (ii) The need for **task generalization** [XNZ18, HNX19] demands that a robot trained on one specific task, such as making a sandwich, should be able to transfer its knowledge to related tasks, like preparing a salad, even when confronted with unfamiliar ingredient layouts or kitchen arrangements.

In robotics and AI literature, **planning** [LaV06] is a fundamental approach that addresses the aforementioned challenges. Under the Markovian assumption [Put90], planning requires defining a state space $\mathcal{S}$, an action space $\mathcal{A}$, a world model $P(s'|s,a)$, and a goal checker $G_g(s)$ or reward model $R_g(s,a)$. Planning then solves for a plan, *i.e.*, a sequence of actions, to reach a goal state s_g or maximize the cumulative reward from an initial state s_0. Ideally, with an appropriate state representation and generalizable world model, planning can solve very long-horizon tasks with arbitrary goals in novel scenarios. For instance, recent Large Language Models (LLMs) [ZZL23], remarkable in understanding natural language and encoding vast world knowledge, can solve diverse planning tasks in the natural language domain when used as world models [HGM23] or

planners [HAP22, HXX23], achieving few-shot or zero-shot generalization. However, additional challenges exist in solving real-world robotics tasks with planning. First, a perception model is required to process raw sensor observations into meaningful state variables, known as "**state abstraction**" or "**scene abstraction**" [KKL18a, CFK22, SHL22]. Additionally, the world model must be defined with an appropriate action space that interfaces with low-level joint commands of robots while being abstract enough to allow effective planning. This perspective is also known as "**action abstraction**" [KKL18a, GCS22, YCT23].

Existing work that tackles robotic tasks with planning falls into two main categories. One seminal formulation is Task and Motion Planning (TAMP) [KL11, GCH21, SFR14, GLK20], which manually designs symbolic state and action abstractions and hierarchically decomposes the planning problem into high-level symbolic task planning and low-level motion planning. The task planning stage searches for long-horizon symbolic action sequences, using hand-crafted planning domains consisting of abstract world knowledge represented in Planning Domain Definition Language (PDDL) [FL03]. The motion planning stage computes feasible motion trajectories subject to geometric constraints, which accomplish each abstract action. TAMP can solve long-horizon robot planning problems with performance guarantees and generalize zero-shot to arbitrary in-domain scenarios and goals. However, this approach requires human experts to manually define abstractions and planning domains [HZZ24, SCK23], limiting its application to diverse domains. Furthermore, TAMP faces significant real-world perception challenges in grounding state symbols and motion planning goals, constraining its applicability to real-world problems [CFK22].

Another line of work is latent-space Model-based Reinforcement Learning (MBRL) [HLF19, HLB19, HWS22], which formulates a holistic framework that learns state variables as latent vectors, along with an encoder that abstracts raw observations into state vectors, a world model that predicts state transitions, and a reward model that implies desired behaviors. The latent state variables are usually learned with additional auxiliary objectives, such as reconstruction loss [HLF19, HLB19] and contrastive loss [HWS22], to preserve nontrivial information. Some works further extend this approach to incorporate hierarchical action spaces [HLF22]. While this

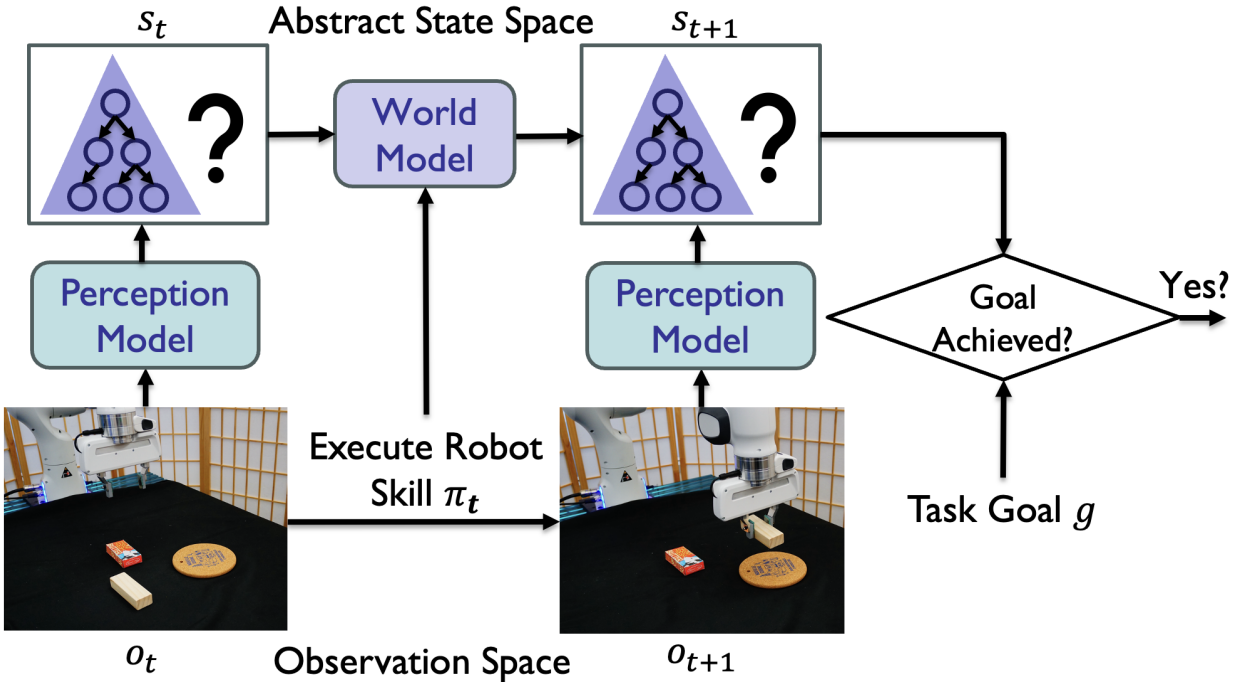


Figure 1.1: **The scene abstraction framework.** Assuming known action abstraction, we acquire a perception model to abstract observations into abstract states, a world model that captures abstract state transition, and a goal checker that verifies whether a goal is reached at an abstract state.

approach learns state representations and world models that directly interface with raw observations and robot actions, it typically works in narrow domains and struggles with generalization. Moreover, the learned neural network-based world model may suffer from compounded prediction errors, which limits the planning horizon.

In this dissertation, we aim to address the limitations of existing approaches and enable generalizable long-horizon robot planning. We consider a general scene abstraction framework depicted in Fig. 1.1. Central to the framework is an abstract state representation that can be perceived from raw observations with a perception model, and a world model that captures abstract state transitions and enables planning in the abstract state space. Such an abstract state representation should preserve important task-relevant information such that: (i) it allows checking whether a task goal is achieved easily, and (ii) it allows a world model to consistently capture abstract state transi-

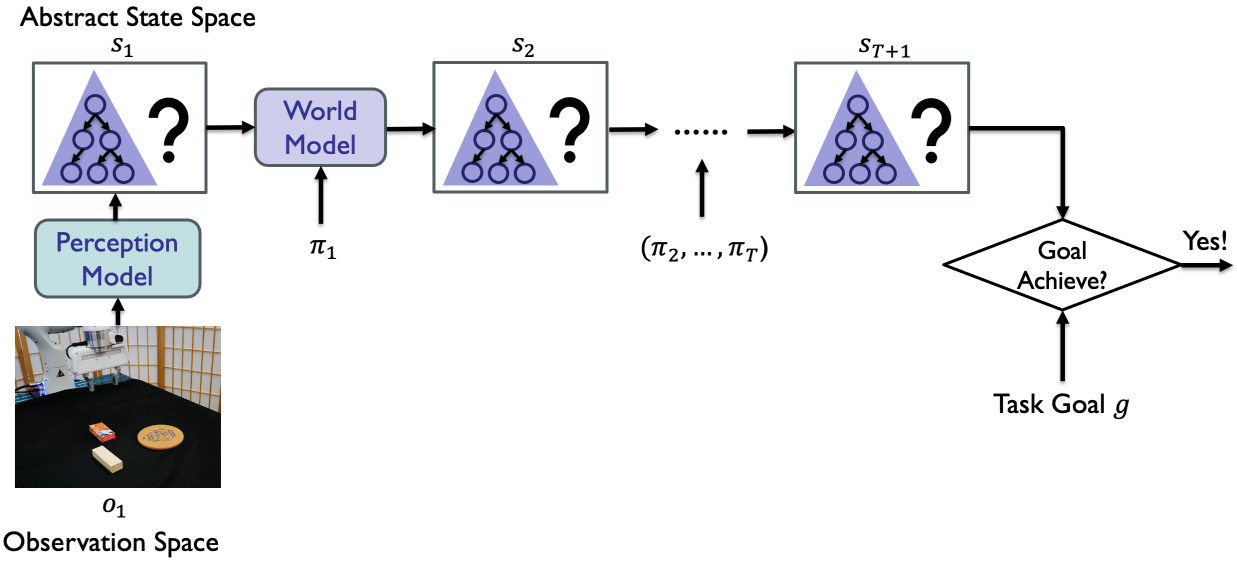


Figure 1.2: **Planning with scene abstraction.** By converting an initial observation into an abstract state, scene abstraction allows effective planning in the abstract state space.

tions on robot actions [KKL18b]. We assume that the robot is equipped with a library of primitive skills $\{\pi(\theta)\}$ each parameterized with discrete object arguments and continuous skill parameters θ [SCK23]. For example, a skill $\texttt{Pick}(a)$ can be used to pick up an object a, while another skill $\texttt{Place}(b, pos)$ places an object b to position pos. These skills establish effective action abstractions and can typically be realized through motion planning, imitation learning, or reinforcement learning. After acquiring the perception model, world model, and goal checker, we can solve complex tasks by first converting the initial observation into an abstract state and then planning skill sequences in the abstract state space by rolling out with the world model (see Fig. 1.2).

In particular, we advocate for scene graph-based representations [ZM07, AHG19, RGA20] that abstract observations into objects and their relations. We show an example of scene graph abstraction in Fig. 1.3. Formally, we define a scene graph as (V, E), with the set of nodes V and the set of directional edges E. Each node $v \in V$ represents an object instance attributed with symbolic object type c and continuous attributes x that describe additional geometric or semantic information, such as object pose. Each edge $e \in E$ represents an ordered and named symbolic relation among objects,

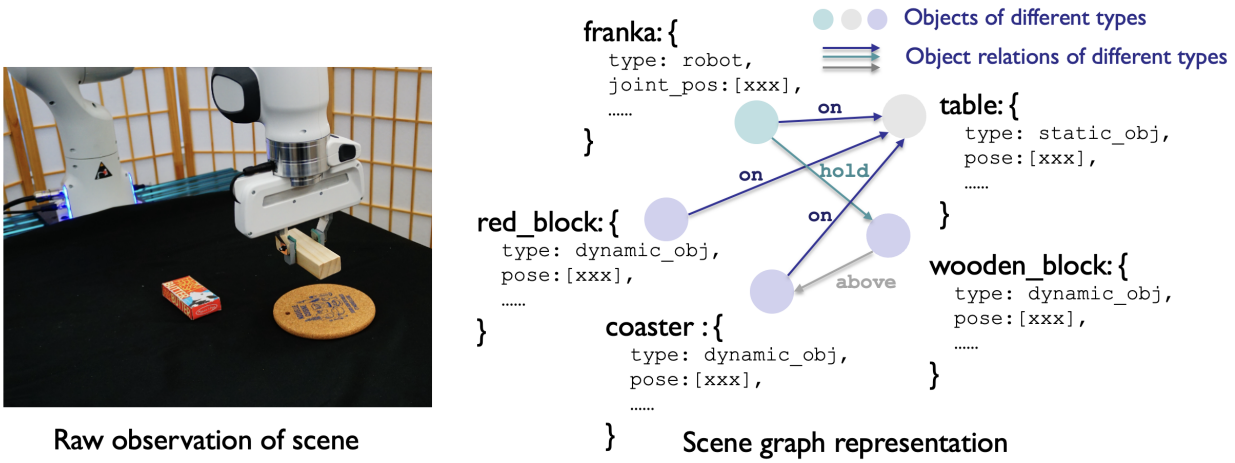


Figure 1.3: **An example of scene graph-based state abstraction.**

such as `on` and `hold`. While Fig. 1.3 only shows binary relations, we allow higher-order relations that involve more than two objects in the scene graph. Such an abstract state representation enjoys various advantages: (i) It abstracts out irrelevant information and spans a **compact** state space, which allows modeling state transitions with a world model with low complexity. This reduces the computation load and enables long-horizon planning. (ii) It is **object-centric** and **relational** and aware of individual objects and disentangled relations, allowing compositional generalization to novel object configurations. We hope that this structural representation unlocks more effective robot planning.

In this dissertation, we present three parts of work that attempt to fill in the missing pieces in the scene abstraction framework in Fig. 1.1 to tackle long-horizon robot planning with generalization:

1. **A robust perception system to build scene graphs from raw observations.** In Chapter 2, we design a contact graph representation that preserves environment kinematics and object supporting relations for robot TAMP. We develop a scene reconstruction system to construct contact graphs of indoor scenes from RGB-D streams. We show that by replacing objects in the contact graph with (potentially articulated) CAD models, the reconstructed scene becomes interactive and affords robot interaction within.

2. **Closed-loop planning frameworks that handle uncertainties and environmental feedback using world model knowledge.** We exploit the internal world knowledge of language models and leverage them as planners to perform open-ended reasoning and planning based on contact graph and other environment feedback. In Chapter 3, we utilize a Vision Language Model (VLM) to reason and plan for a series of real-world mobile manipulation tasks based on multi-modal feedback of contact graph and camera image. In Chapter 4, we explore using an Large Language Model (LLM) to propose symbolic plans and action parameters for TAMP, reacting to the contact graph and detailed feedback from a motion planner.

3. **Data-efficient approaches to learn object and relational symbols in scene graphs and world models with minimal human intervention.** We propose to jointly learn symbols and world models from interaction data. In Chapter 5, we present an interaction framework for robots to learn relational symbols, *i.e.*, predicates, together with world model, *i.e.*, operators, from interaction data and human language feedback. The learned predicates and operators are compiled into a PDDL domain file to enable search-based task planning. In Chapter 6, we study learning symbolic object types in an object cutting task, which involves handling object fragments without semantic labels. We propose a probabilistic framework to cluster object shape features into discrete types and induce a grammar-based world model to capture state transitions. In both works, we show the learned symbolic representation and world model enable compositional generalization to novel objects and goals.

Part I

Perception: Scene Graph Reconstruction for Robot Interaction

CHAPTER 2

Scene Reconstruction with Contact Graph for Robot Interaction

This chapter rethinks scene reconstruction from an embodied agent's perspective: While the classic view focuses on the reconstruction accuracy, our new perspective emphasizes the underlying functions and constraints of the reconstructed scenes that provide *actionable* information for robot *interactions*. We first design a contact graph representation that hierarchically organizes functional objects and contexture relations (*e.g.*, supporting, proximal), which preserves kinematic information of the environment and supports task and motion planning in the scene. Then we develop a perception system that reconstructs contact graphs from RGB-D streams, and further replaces reconstructed objects with potentially articulated CAD models. The system produces *functionally equivalent* and interactive scenes that afford finer-grained robot interactions in simulation. The materials in this chapter have been published in [HZJ21, HZJ22].

2.1 Introduction

Perception of man-made environments and the objects within inevitably leads to the course of actions [Gib50, Gib66], which naturally form the basis for a human agent to interact with the environment and accomplish complex tasks. Crucially, what we "see" is much more than pixels and semantic labels [KR96]. Instead, we further "see" *how* to interact with them for our task purposes. Likewise, an embodied AI agent or a robot must possess a similar perceptual capability to achieve a wide range of task goals in the physical world. However, this critical perspective is mostly

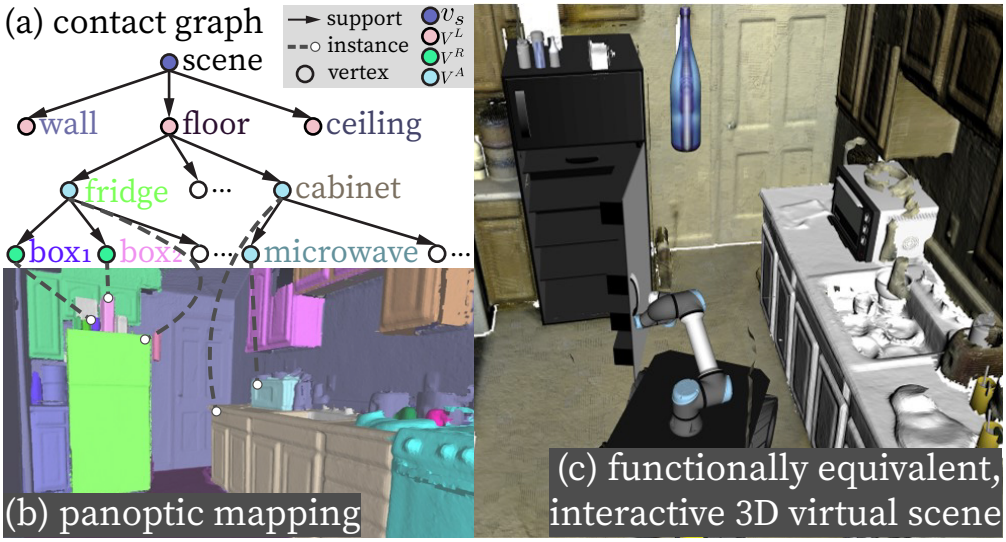


Figure 2.1: **Reconstruction of a functionally equivalent, interactive 3D scene.** (a) A contact graph representation emerged from (b) panoptic mapping. After replacing objects with CAD models, (c) the functionally equivalent, interactive scene enables simulating robot interactions.

unexplored by the prior scene reconstruction literature in computer vision or Simultaneous Localization and Mapping (SLAM) methods in robotics. Oftentimes, prior art only captures scenes' occupancy information and are evaluated primarily by reconstruction accuracy in the Euclidean space. Without incorporating the *actionable* information—actions a semantic entity could afford and the associated physical constraints among entities—in a reconstructed scene, a robot can only perform relatively simple navigation or pick-and-place tasks, hindering its capability in planning and executing tasks with a long horizon.

Take the scene in Fig. 2.1 as an example, where the robot is tasked to pick up a frozen meal from the fridge, microwave it, and serve it. The challenges of processing *actionable* information are three-fold. First, it needs to recognize the semantics and geometry information of objects (*e.g.*, this piece of point cloud is a fridge). Although typical semantic mapping and segmentation techniques can achieve this goal [HLS20, NSI19], a more robust and accurate approach is still in need to better handle the complexity in clustered real environments given a first-person-view RGB-D video stream. Second, mere semantics are inadequate to reflect the actions an object

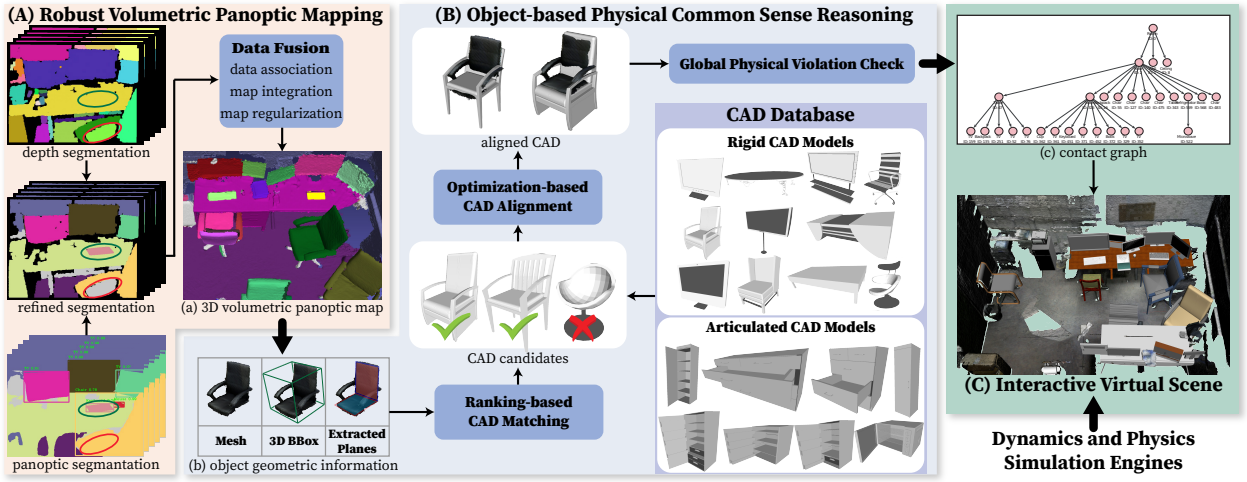


Figure 2.2: **System architecture.** (A) Per-frame segmentation and global data fusion produce (a) a 3D volumetric panoptic map with fine-grained semantics and geometry, served as the input for (B) physical common sense reasoning that matches and aligns segmented object meshes with functionally equivalent CAD models. Specifically, (b) by geometric similarity, a ranking-based matching algorithm selects a shortlist of CAD candidates, followed by an optimization-based process that finds a proper transformation and scaling between CAD candidates and object mesh. A global physical violation check is applied to finalize CAD replacements to ensure physical plausibility. (C) This CAD augmented scene can be imported to existing simulators; (c) contact graph encodes kinematic relations among scene entities and reflects the planning space for a robot.

affords (*e.g.*, whether or how the fridge can be opened). While some existing work attempted to identify the associations between symbolic actions and objects [MTF15, LLK19] or the underlying the object's kinematics [SSB11, CD17, MB19], they are insufficient for robots to execute complex tasks with multiple steps at the motion level. Third, we quest for a more fundamental question: How to devise a scene representation with a succinct action specification and task definition to account for the action opportunities and the accumulated outcome of executed actions. Without addressing these challenges, a robot can hardly plan for the given task or verify whether its plan is valid before executing in the physical world.

In this chapter, we propose a new task of reconstructing *functionally equivalent* and interactive

scenes by representing the *actionable* information of scene entities to support agents' planning and simulation. Here we argue that a scene's functionality is composed by the functions of objects within the scene. Therefore, the essence of a *functionally equivalent* scene is to preserve most objects' four characteristics with a decreasing propriety: (i) their semantic class and spatial relations with nearby objects, (ii) their affordance, *e.g*. what interactions they offer, (iii) similar geometry in terms of size and shape, and (iv) similar appearance. To address this new task, we devise a perception system with three unique components; see an illustration in Fig. 2.2:

A) A robust 3D volumetric panoptic mapping module, detailed in Section 2.3, accurately segments and reconstructs 3D objects and layouts in clustered scenes based on potentially noisy per-frame segmentation. The term "panoptic," introduced in [KHG19], refers to jointly segmenting *stuff* and *things* in semantic and instance levels. In this work, we regard objects as *things* and layouts as *stuff*. This module produces a volumetric panoptic map using a novel per-frame panoptic fusion strategy and a global data fusion procedure performing data association, map integration and regularization; see Fig. 2.1b and Fig. 2.2a for examples of results.

B) A physical reasoning module, detailed in Section 2.4, replaces the potentially noisy and incomplete object meshes segmented from the panoptic map with functional (rigid or articulated) CAD models. This step is achieved by a ranking-based CAD matching and an optimization-based CAD alignment, which accounts for both geometric and physical constraints. We further introduce a global physical violation check to ensure that the resulting reconstructed interactive scene is physically plausible.

C) A contact graph cg representation, detailed in Section 2.2 and illustrated in Fig. 2.3, is constructed in accordance with the supporting and proximal relations among objects and imposes physical constraints as well as kinematic information for a robot's task execution. After retrieving *actionable* information annotated in CAD models, this novel representation indicates how an object can be moved or manipulated (*e.g*., a table can be moved in 3D space) and how nearby objects would move correspondingly (*e.g*., a box on the table would go through a similar transformation if not slid or tilted). The cg can be interpreted as and converted to a kinematic tree, which is updated

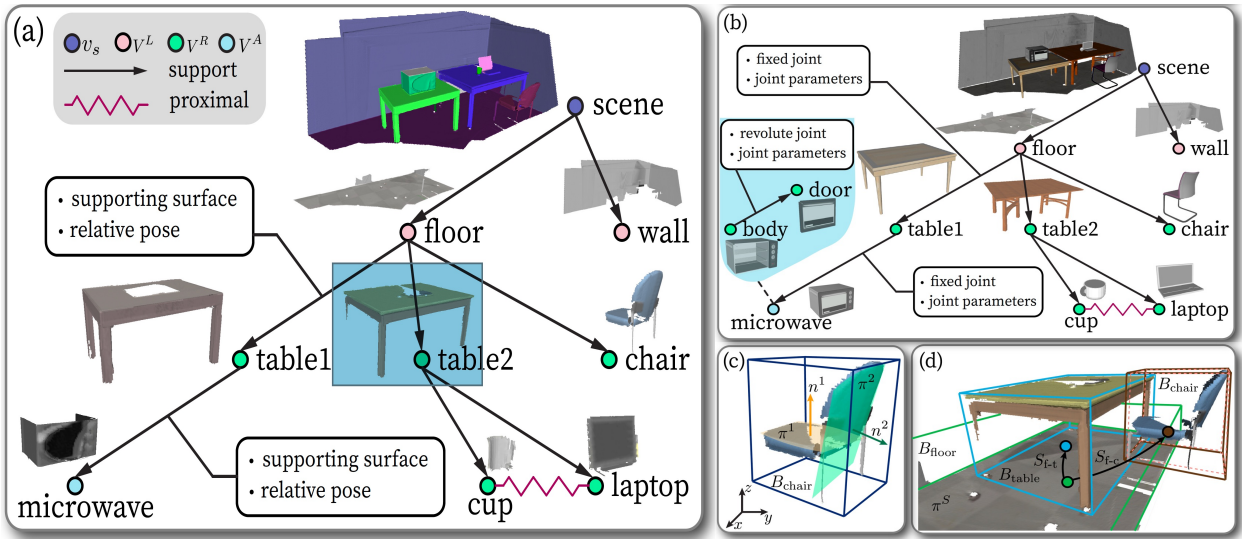


Figure 2.3: **Contact graph and relations within.** Each node denotes an object or a piece of layout, reconstructed and segmented as meshes from the RGB-D stream using the proposed panoptic mapping module. The directed edges indicate supporting relations—The parent node supports the child node. (b) The object meshes are replaced by best-fitted CAD models to create a functionally equivalent and physically plausible reconstructed scene. The directed edges and the constructed kinematic relations define the action space for robot planning. (c) The supporting relations can refine (d) the 3D bounding box estimation. Initial: dashed line. Refined: solid line.

following the robot's actions to support long-horizon task and motion planning. As such, it serves as an ideal representation that bridges robot perception (scene reconstruction) with robot planning.

2.1.1 Related Work

Modern **semantic mapping** [NSI19, GFN19, PHN19] and **object SLAM** [YS19a, MCB18] methods can retrieve object semantic segmentation, 6 DoF poses, and 3D bounding boxes during reconstruction. Physical cues, such as support and collision [YS19b, WSJ20, SCX20] and robot proactive actions [XHS15, LXS18], can be further integrated to better estimate and refine the scene semantics. In parallel, significant efforts have been made for object instance segmentation

from point clouds [ZZC19]; *e.g.*, [YZW19] can segment an object with fine-grained part instances, and [PNH19] jointly perform semantic and instance segmentation. The above work, however, could only produce incomplete objects (in contrast to full 3D) due to confined viewpoints in the physical world, which prohibits the complex robot interaction and task execution in the reconstructed scenes. To alleviate this issue, researchers have recently attempted to **align CAD models** to these incomplete objects based on single RGB image [HQZ18, CHY19], single RGB-D image pair [GAG15, ZGL19], and scanned scene meshes [DCS17, ADD19, ADN19] to incorporate richer scene semantics. Following this trend, our system further aligns (part-based) CAD models to segmented objects to enable robot manipulation and interaction.

Devising an appropriate **scene representation** for scene reconstruction remains an open problem [CCC16]. Existing SLAM and semantic mapping approaches reviewed above oftentimes represent a reconstructed scene and its entities as sparse landmarks [PJ12, YS19a], surfels [MHD17, HLS20], volumetric voxels [GFN19, MCB18], or semantic objects [YS19a, MCB18]. Such a paradigm only provides geo-information of what and where to a robot without any actionable information for its interactions or planning. Meanwhile, graph-based representations for 3D scene further identify the hierarchical and relational structure among the scene entities [ZM07, ZZ11, ZZ13, ZZY15, HQX18, JQZ18, CHY19, AHG19, WDN20, RGA20], providing better structural and contextual information of the reconstructed scenes. In particular, [RGA20] explicitly incorporate actionable information to support robot planning, though limited to navigation and traversal tasks as the representation only models the connectivity between entity nodes. [RGA20] is also limited in that it is conducted in a simulated environment without accounting for real perception challenges. By leveraging the advantages of prior arts and addressing the shortcomings, the proposed system takes a real RGB-D stream as input and produces a contact graph representation based on the identified supporting relations among scene entities. This representation for scene reconstruction indicates how an entity can be interacted with and what the effect would be after an interaction, capable of supporting more complex manipulation planning.

2.1.2 Contributions

To our knowledge, ours is the first work that introduces a comprehensive system that reconstructs a full 3D scene from an embodied agent's perspective to provide *actionable* information for simulating robot interactions. It makes three major contributions:

1. We introduce a novel scene representation using a contact graph, whose structure is determined by the supporting and proximal relations among scene entities. It imposes physical constraints for a physically plausible scene and kinematic information that indicates whether and how an object can be interacted with. This contact graph representation is constructed and maintained for the scene reconstruction, and converted to a kinematic tree, which reflects the full geometric state of a scene and updates to keep track of every interaction. As such, our contact graph representation can facilitate the functionally equivalent scene reconstruction, as well as the robot learning and planning for complex long-horizon tasks.
2. Leveraging (i) local geometric similarity on the basis of relative sizes and surfaces of each object, and (ii) global physical constraints regarding the plausibility of stable support and non-penetration, we align rigid or articulated CAD models to object meshes to generate a physically plausible, fully interactive scene.

2.2 Contact Graph Representation

We devise a graph-based representation, *contact graph* cg, to represent a 3D indoor scene and the relations among scene entities. Formally, a contact graph $cg = (pt, E)$ contains (i) a parse tree (*pt*) that hierarchically organizes the scene entities [ZM07], and (ii) the proximal relations E among entities represented by undirected edges; see an example in Fig. 2.3a.

2.2.1 Representation

Scene Parse Tree $pt = (V, S)$ has been used to represent the hierarchical decompositional relations (*i.e.*, the edge set S) among entities (*i.e.*, the node set V) in various task domains, including 2D images and 3D scenes [ZM07, ZZ11, ZZ13, QZH18, JQZ18, HQZ18, HQX18, CHY19], videos and activities [ZZZ15, ZJZ16, QJH20, JCH20], robot manipulations [EGX17, LZS18, EGL19, LZZ19, ZZZ20], and theory of mind [YLF20]. In this work, we adapt *pt* to represent supporting relations among entities instead of their decomposition. A *pt* is dynamically built and maintained during the reconstruction based on the identified supporting relations among segmented scene entities; for instance in Fig. 2.3a, the `table1` is the parent node of the `microwave`. Supporting relation is quintessential in scene understanding as it reflects the omnipresent physical plausibility; *i.e.*, if the table were moved, the microwave would move together with it. This perspective of physical common sense goes beyond occupancy information (*i.e.*, the geometric location of an object); in effect, it further provides actionable information and the potential outcome of actions for robot interactions and task executions in the scene.

Scene Entity Nodes $V = \{v_s\} \cup V^L \cup V^R \cup V^A$ include: (i) the scene node v_s, serving as the root of pt, (ii) layout node set V^L, including floor, ceiling, and the walls that bound the 3D scene, (iii) rigid object set V^R, wherein each object has no articulated part (*e.g.*, a table), and (iv) articulated object set V^A, wherein each object has articulated parts to be interacted for robot tasks (*e.g.*, fridge, microwave). Each non-root node $v_i = \langle o_i, c_i, M_i, B_i(\boldsymbol{p}_i, \boldsymbol{q}_i, \boldsymbol{s}_i), \Pi_i \rangle$ encodes a unique instance label o_i, a semantic label c_i, a full geometry model M_i (*e.g.*, a triangle mesh or a CAD model), a 3D bounding box B_i (parameterized by its center position $\boldsymbol{p}_i$, orientation $\boldsymbol{q}_i$, and size $\boldsymbol{s}_i$, all in $\mathbb{R}^3$), and a set of surface planes $\Pi_i = \{\boldsymbol{\pi}_i^k, k = 1 \cdots |\Pi_i|\}$, where a plane $\boldsymbol{\pi}_i^k$ is represented by a homogeneous vector $[\boldsymbol{n}_i^{kT}, d_i^k]^T \in \mathbb{R}^4$ in the projective space [HZ03] with unit plane normal vector $\boldsymbol{n}_i^k$, where any point $\boldsymbol{v} \in \mathbb{R}^3$ on the plane satisfies a constraint: $\boldsymbol{n}_i^{kT} \cdot \boldsymbol{v} + d_i^k = 0$; see Fig. 2.3c for an illustration. Compared to other geometric primitives like generalized cylinders, planes are advantageous in that they can be extracted robustly from corrupted object meshes and are effective features in downstream computations.

Supporting Relations S is the set of directed edges in pt from parent nodes to their child nodes. Each edge $s_{p,c} \in S$ imposes physical common sense between the parent node v_p and the child node v_c. These constraints are necessary to ensure that v_p supports v_c in a physically plausible fashion:

(1) Geometrical plausibility. The parent node v_p should have a plane $\boldsymbol{\pi}_p^s = [\boldsymbol{n}_p^{sT}, d_p^s]^T$ that is horizontal and is in contact with the bottom surface of the child v_c:

$$\begin{aligned} &\exists \boldsymbol{\pi}_p^s \in \Pi_p, \boldsymbol{n}_p^{sT} \cdot \boldsymbol{g} \leqslant a_{th}, \\ &s.t.\ \mathcal{D}(v_c, \boldsymbol{\pi}_p^s) = p_c^g - (-d_p^s + s_c^g/2) = 0, \end{aligned} \tag{2.1}$$

where $\boldsymbol{g}$ is a unit vector in the gravity direction, $a_{th} = -0.9$ is a tolerance coefficient ($a_{th} = -1$ for a perfect horizontal plane), and p_c^g and s_c^g denote the position and size of the v_c's 3D bounding box along the gravity direction, respectively.

(2) Sufficient contact area for stable support. Formally,

$$\mathcal{A}(v_p, v_c) = \mathrm{A}(v_p \cap v_c)/\mathrm{A}(v_c) \geqslant b_{th}, \tag{2.2}$$

where $\mathrm{A}(v_c)$ is the bottom surface of the v_c's 3D bounding box, and $\mathrm{A}(v_p \cap v_c)$ is the area of the overlapping rectangle containing the mesh vertices of v_p near $\boldsymbol{\pi}_p^s$ within v_c's 3D bounding box. We set threshold $b_{th} = 0.5$ for a stable support.

Proximal Relations E introduce links among entities in the *pt*. It imposes additional constraints by modeling spatial relations between two non-supporting but physically nearby objects v_1 and v_2: Their meshes should not penetrate with each other, *i.e.*, $\mathrm{Vol}(M_1 \cap M_2) = 0$. Note that we only assign a proximal relation between two objects with overlapping 3D bounding boxes, *i.e.*, when $\mathrm{Vol}(B_1 \cap B_2) > 0$, instead of between every pair of objects to reduce computation cost. The non-penetration constraints will be applied when selecting physically plausible scene configurations, as detailed in Section 2.4.4.

2.2.2 Constructing Contact Graphs

For each scene entity x extracted from the volumetric panoptic map (see details on obtaining panoptic map in Section 2.3.4), we initialize a scene entity node v_x of cg by: (i) acquiring its o_x, c_x, M_x from the panoptic map, (ii) estimating a gravity-aligned, minimal 3D bounding box $B_x(\boldsymbol{p}_x, \boldsymbol{q}_x, \boldsymbol{s}_x)$ based on M_x using the method in [MB02], (iii) detecting a set of surface planes Π_x on M_x by iteratively applying RANSAC [TJR13] and removing plane inliers. We further classify each initialized scene entity node v_x as a layout node, a rigid object node, or an articulated object node based on its semantic class c_x.

Given a set of scene entity nodes initialized on-the-fly, we apply a bottom-up process to build up the structure of cg by estimating supporting relations among the entities. Specifically, for each node v_c, we find a parent node v_p with a supporting plane $\boldsymbol{\pi}_p^s$ that best satisfies the constraints described in Eqs. (2.1) and (2.2). We consider all nodes $\{v_i\}$ whose bottom planes are spatially below the 3D bounding box of v_c as v_p candidates, and acquire their gravity-opposed surface planes $\{\boldsymbol{\pi}_i^k\}$ as potential supporting planes. Then the most likely supporting relation is determined by maximizing the following score function:

$$S(v_c, v_i, \boldsymbol{\pi}_i^k) = \left\{1 - \min\left[1, \|\mathcal{D}(v_c, \boldsymbol{\pi}_i^k)\|\right]\right\} \times \mathcal{A}(v_i, v_c), \tag{2.3}$$

where the first term indicates the alignment between the v_c's bottom surface and the supporting plane, and the second term reflects an effective supporting area, both normalized to $[0, 1]$. We may also uncover an invisible supporting plane (*e.g.*, a fully occluded tabletop). When v_c is well-overlapped with v_i but v_i has no valid supporting plane, the bottom plane of v_c with be registered as a new supporting plane of v_i. This advantage is however hard to guarantee at all time due to the complexity of real-world scenarios. Finally, we construct cg and assign the attributes for each supporting edge based on the estimated supporting relations.

We further refine the 3D bounding box B_i of each scene entity node v_i such that Eq. (2.1) is strictly satisfied and the cg is feasible. This step also compensates for the error of extracting geometric features directly from incomplete reconstructed mesh. Fig. 2.3d illustrates an example

of the refinement process. The reconstructed scene only produces a partial mesh of the chair; its legs are captured incompletely. Consequently, its 3D bounding box (in dashed line) only encloses the detected portion of the chair, which is floating in the air. By determining the supporting relation between the floor and the chair, our system automatically extends the bounding box (in solid line) to the supporting plane on the floor, thus reconstructed a physically plausible scene. In experiments, we also quantitatively evaluate this refinement process; see the result in Table 2.4. As the last step of cg construction, we determine the proximal relations by comparing pairwise 3D bounding boxes of scene entities.

2.2.3 Interpreting a Contact Graph

As shown in Fig. 2.3a and described above, a cg hierarchically organizes segmented scene entities with corresponding semantics, meshes, and extracted geometric features. To convey richer *actionable* information, we convert the cg to a functionally equivalent cg' by maintaining the overall graph structure and replacing each object mesh with a CAD model while preserving its semantic class, instance label, relative dimension, and surface planes; see Fig. 2.3b.

The functionally equivalent cg' with CAD models naturally encodes the full (detected) geometry state of the scene. It can be interpreted as a kinematic tree, where nodes represent links, and edges represent joints connecting two links with assumed joint type, range, and joint value. Depending on the semantic class, individual objects may be replaced by articulated CAD models. For instance, the CAD model for the microwave in Fig. 2.3b consists of two parts, the body and the door, connected by a revolute joint. The cg' (the kinematic tree) is an ideal representation to support robot planning; its joint specifications reflect the possible ways a robot can change environment states and naturally define the task goal for a robot to achieve. Although the knowledge of the object structure is injected when designing the CAD model and is not likely to match with the real one strictly, it nevertheless provides an approximation for most of the possible actions an agent can take and what the actions like, sufficient for the agent's long-term planning.

2.3 Robust Panoptic Mapping

Robust and accurate mapping of scene entities and segmenting them from clustered environments are essential for constructing a cg and serving our downstream tasks. We develop a robust 3D panoptic mapping module to generate object and layout segments in the form of meshes from RGB-D streams; see the pipeline in Fig. 2.2A. Based on the architecture of Voxblox++ [GFN19], our mapping module incorporates crucial modifications to improve the robustness of mapping against noisy and inconsistent segmentation at each frame.

Voxblox++ [GFN19] builds a volumetric object-centric semantic map by (i) generating per-frame segments in point cloud form by combining RGB-based instance segmentation and depth-based geometric segmentation, and (ii) associating the segments across different frames and integrating them into a Truncated Signed Distance Field (TSDF)-based object-level global map. Each per-frame segment is obtained by assigning a semantic label and an instance label produced by instance segmentation to a geometric segment produced by geometric segmentation. Assuming that segments computed using geometry cues are consistent across different frames, Voxblox++ [GFN19] associates those per-frame segments from different views with global map segments by their 3D overlapping ratio and integrates them into the global map, while recording the history of predicted semantic and instance labels for each global map segment.

However, we observe two major limitations of the Voxblox++ [GFN19]. First, the generated per-frame segments may not preserve all predicted instances and some segments of far-away background may be labeled as foreground objects, negatively affecting the mapping performance. We design two extra steps to handle this limitation, as detailed in Section 2.3.1. Second, Voxblox++ separately tracks semantic and instance labels in data association and map integration processes, making it less coherent when identifying instance and recognizing semantics for the same global map segment. Our solution is to jointly account for semantic and instance labels throughout the procedure to build a more consistent global map. We describe our implementation of this strategy in data association (Section 2.3.2), map integration and regularization (Section 2.3.3), and scene

entity extraction (Section 2.3.4).

2.3.1 Per-frame Segmentation and Fusion

Following Voxblox++ [GFN19], we perform RGB-based panoptic segmentation and depth-based geometric segmentation for each frame and then combine the two sets of segments. Given a RGB-D image as the input, we use an off-the-shelf panoptic segmentation tool provided by Detectron2 [WKM19] to produce panoptic segments in RGB domain. A convexity-based depth segmentation approach [FNF18] can segment the corresponding depth image following geometric boundaries. We denote each predicted 2D panoptic segment as M_i with semantic label c_i and instance label o_i (whereas each stuff class has only one instance label), and each 3D geometric segment (in point cloud) as G_j. Then the goal is to fuse the segmentation from two sources to generate per-frame point cloud segments $\{(P_k, c_k, o_k)\}$, which preserve the predicted geometric and semantic information.

Voxblox++ [GFN19] generates $\{(P_k, c_k, o_k)\}$ by assigning semantic and instance labels to geometric segments $\{G_j\}$ greedily based on the 2D overlap between the 2D projection of each G_j and $\{M_i\}$ on the image coordinate. In practice, this strategy leads to two drawbacks. The first one is that predicted instances will be ignored if they are not recognized geometrically in depth images. Fig. 2.2A shows an example, the missing keyboard marked by a green circle in depth segmentation would be discarded by Voxblox++. We instead split a geometric segment G_j to extract the point cloud corresponding to a panoptic segment M_i if the 2D projection of G_j fully contains M_i when aligned. Then we assign semantic and instance labels for all G_j as well as the extracted point cloud segments as [GFN19] does to get $\{(P_k, c_k, o_k)\}$. Secondly, an inaccurately segmented object in RGB image may consist of far-away geometric segments in depth, *e.g.*, the floor marked by a red circle is regarded as part of the chair in the panoptic segmentation in Fig. 2.2A. Our modification addresses this issue by adding an extra step of Euclidean clustering. We compute pairwise Euclidean distances among all geometric segments that belong to the same object instance, and applying Euclidean clustering to obtain clusters of segments. Then we retrieve the largest clus-

ter defined as having the largest total number of points in its segments, and keep the segments within as part of the instance. The rest of segments are regarded as outliers and assigned to the background.

The above implementation relies on some defined heuristics that could limit the generalizability of our panoptic segmentation approach; one direction to overcome this limitation is to introduce data-driven methods, which is beyond the scope of the work. Nevertheless, the two proposed steps are useful practice that significantly improves the per-frame segmentation. As an example shown in Fig. 2.2a, our method (i) correctly segments the keyboard and divides the two monitors when they are geometrically under-segmented, (ii) obtains geometrically refined panoptic segmentation of the table, chair, and floor, and (iii) excludes the far-away ground from the segmentation of the chair.

2.3.2 Data Association

We associate each per-frame point cloud segment to a global 3D segment (or global segment for short) in the global map, while associating its panoptic prediction with a global panoptic entity. Note that the global segments and panoptic entities are maintained and updated throughout the entire mapping process. Following Voxblox++ [GFN19], we first draw the correspondence between per-frame segments and global segments greedily based on their 3D overlaps given the camera trajectory. We denote that each global segment is indexed with a unique segment label $l \in \mathbb{L}$.

For each per-frame segment (P_k, c_k, o_k) associated with a global segment l_i, we aim to find its associated global instance label p_m by looking at the past panoptic predictions of segment l_i. We introduce a triple-wise count $\Phi(l, c, p)$ over a segment label l, a semantic label c, and an instance label p in the global map to jointly track the semantic and instance predictions. This is inspired by the observation that the prediction of instances and their semantic labels are inter-dependent in typical object detection and segmentation algorithms [RHG16, HGD17]. Specifically, p_m is assigned with the instance label p that maximizes the count $\Phi(l_i, c_k, p) > 0$. When $\sum_p \Phi(l_i, c_k, p) = 0$, we

assign a new global instance label $p_m = p_{new}$. We further prevent assigning multiple labels with the segments that have the same instance labels.

2.3.3 Map Integration and Regularization

We integrate per-frame segments into the 3D volumetric panoptic map by (i) integrating the segments into a TSDF volume [OTF17] with each TSDF voxel labeled with a global segment label l, and (ii) recording the associated panoptic entities. For any per-frame segment associated with (l_i, c_k, p_m), we increase the triple-wise count:

$$\Phi(l_i, c_k, p) = \Phi(l_i, c_k, p) + 1. \tag{2.4}$$

We also introduce a two-stage process to regulate the map by merging global segment labels and instance labels. Specifically, we first merge global segment labels pairwise if they share voxels over a certain ratio [GFN19]. Next, we merge two global instance labels $p_1, p_2 \in \mathbb{P}$ with the same semantic class $c \in \mathbb{C}$ if the duration of association with common segment labels exceeds a threshold:

$$\sum_{l \in \mathbb{L}_\cap} \left[\Phi(l, c, p_1) + \Phi(l, c, p_2)\right] \geqslant m_{th} \cdot \sum_{l \in \mathbb{L}} \left[\Phi(l, c, p_1) + \Phi(l, c, p_2)\right], \tag{2.5}$$

where $\mathbb{L}_\cap = \{l \in \mathbb{L} | \Phi(l, c, p_1) > 0, \Phi(l, c, p_2) > 0\}$. This step merges incorrectly split instances, which can be introduced by the overcautious filtering step when generating per-frame point cloud segments. We note that this map regularization process can be regarded as a delayed data association that corrects potentially wrong association of global segments and instances. It helps improve the consistency and scalability of the global map; *i.e.*, it reduces the map size.

2.3.4 Panoptic Entities Extraction

After the above mapping process, we extract the panoptic entities (*i.e.*, objects and layouts) from the global map as triangle meshes. For each global segment l, its semantic class $\hat{c}_l$ and global

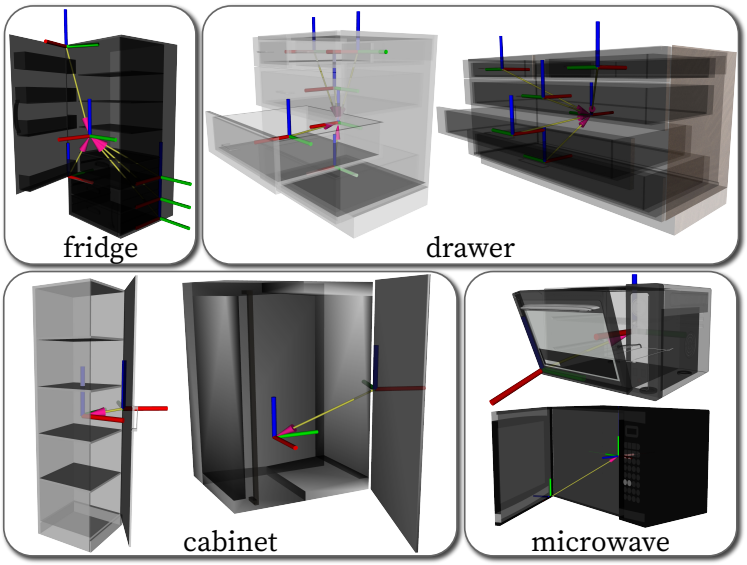


Figure 2.4: **Examples of articulated CAD models in the database.**

instance label $\hat{p}_l$ are determined following a greedy strategy:

$$\begin{aligned} \hat{c}_l &= \arg\max_{c \in \mathbb{C}} \sum_{p \in \mathbb{P}} \Phi(l, c, p), \\ \hat{p}_l &= \arg\max_{p \in \mathbb{P}} \Phi(l, \hat{c}_l, p). \end{aligned} \tag{2.6}$$

For each global instance label $p \in \mathbb{P}$, we group all global segments in the map with labels in the set $L_p = \{l \in \mathbb{L} | \hat{p}_l = p\}$ and extract the corresponding TSDF volume, from which a mesh is created. In a nutshell, our system outputs a set of scene entities in the form of triangle meshes with their instance labels and semantic labels.

2.4 Scene Reconstruction with CAD Replacement

Due to occlusion or limited camera angle, the reconstructed scene and the segment meshes are oftentimes incomplete and non-interactive before recovering them as full 3D models; Fig. 2.5a and Fig. 2.6a show some examples of incomplete meshes. We introduce a multi-stage framework to replace a segmented object mesh with a CAD model through (i) an object-level CAD matching, (ii) pose alignment of the CAD model, and (iii) a scene-level, global physical violation check; see

Fig. 2.2B for an illustration of the framework.

2.4.1 CAD Pre-processing

We collect a CAD database consisting of both rigid and articulated CAD models, organized by semantic classes. The rigid CAD models are obtained from ShapeNetSem [CFG15], whereas articulated ones are first assembled and then properly transformed into one model. Each CAD model is transformed to have its origin and axes aligned with its canonical pose. Fig. 2.2B shows some instances of CAD models in the database, and Fig. 2.4 highlights some articulated CAD examples with coordinate frames on the articulated parts. All the objects can be uniformly scaled while persevering transformation and kinematic information for the subsequent matching and alignment. Similar to a segmented scene entity x, a CAD model y is parameterized by o_y, c_y, M_y, while we further extract its $B_y(\boldsymbol{p}_y, \boldsymbol{q}_y, \boldsymbol{s}_y)$, and Π_y.

2.4.2 Ranking-based CAD Matching

Take the chair in Fig. 2.2b as an example: Given a segmented object entity x, the algorithm retrieves all CAD models in the same semantic category (*i.e.*, chair) from the CAD database to best fit x's geometric information. Since the exact orientation of x is unknown at this step yet, we uniformly discretize the orientation space into 24 possible orientations. For each rotated CAD model y that aligned to one of the 24 orientations, the algorithm computes a Matching Error (ME):

$$D(x, y) = \omega_1 \cdot d_s(x, y) + \omega_2 \cdot d_\pi(x, y) + \omega_3 \cdot d_b(y), \tag{2.7}$$

where $\omega_1 = \omega_2 = 1.0$ and $\omega_3 = 0.2$ are the weights of three terms, set empirically. We detail these terms below.

(1) d_s computes the difference of relative 3D bounding boxes sizes between the segmented mesh and the CAD model:

$$d_s(x, y) = \left\| \frac{\boldsymbol{s}_x}{\|\boldsymbol{s}_x\|_2} - \frac{\boldsymbol{s}_y}{\|\boldsymbol{s}_y\|_2} \right\|. \tag{2.8}$$

(2) d_π penalizes the misalignment between their surface planes in terms of plane normal and relative distance:

$$d_\pi(x,y) = \min_{f_\Pi} \sum_{\boldsymbol{\pi}_i \in \Pi_x} \left[\left\| \frac{d({T_x}^T \boldsymbol{\pi}_i)}{\|\boldsymbol{s}_x\|_2} - \frac{d(f_\Pi(\boldsymbol{\pi}_i))}{\|\boldsymbol{s}_y\|_2} \right\| + 1 - \boldsymbol{n}(\boldsymbol{\pi}_i)^T \cdot \boldsymbol{n}(f_\Pi(\boldsymbol{\pi}_i)) \right], \tag{2.9}$$

where T_x denotes the homogeneous transformation matrix from the map frame on the ground to the frame of the bounding box B_x, $d(\cdot)$ the offset of a plane, $\boldsymbol{n}(\cdot)$ the normal vector of a plane, and $f_\Pi : \Pi_x \rightarrow \Pi_y$ a bijection function denoting the assignment of feature planes between x and y. Note that f_Π is also constrained to preserve supporting planes as defined in Eq. (2.1). As computing d_π involves solving an optimal assignment problem, we adopt a variant of the Hungarian algorithm [JV87] to identify the best f_Π between the set of surfaces extracted from a segmented object mesh and that from a candidate CAD model. Then we can calculate the misalignment error term $d_\pi(x,y)$ that candidate CAD introduces.

(3) $d_b(y)$ is a bias term that adjusts the overall matching error for less preferable CAD candidates:

$$d_b(y) = 1 + \boldsymbol{g}^T \cdot \boldsymbol{z}(y), \tag{2.10}$$

where $\boldsymbol{z}(y)$ denotes the up-direction of the CAD model in the oriented CAD frame, and $\boldsymbol{g}$ is a unit vector along the gravity direction. Generally, we prefer CAD candidates that are upright instead of leaning aside.

Fig. 2.5b illustrates the matching process. Empirically, we observe that the discarded CAD candidates of "chair" and "table" due to large Matching Error (ME) are indeed more visually distinct from the segmented object meshes. Moreover, the "fridge" model with a wrong orientation leads to a much larger ME and is thus discarded. These results demonstrate that our ranking-based matching process can select visually more similar CAD models with a roughly correct orientation. Our system maintains the top 10 orientated CAD candidates with the lowest ME for more accurate alignment in the next stage.

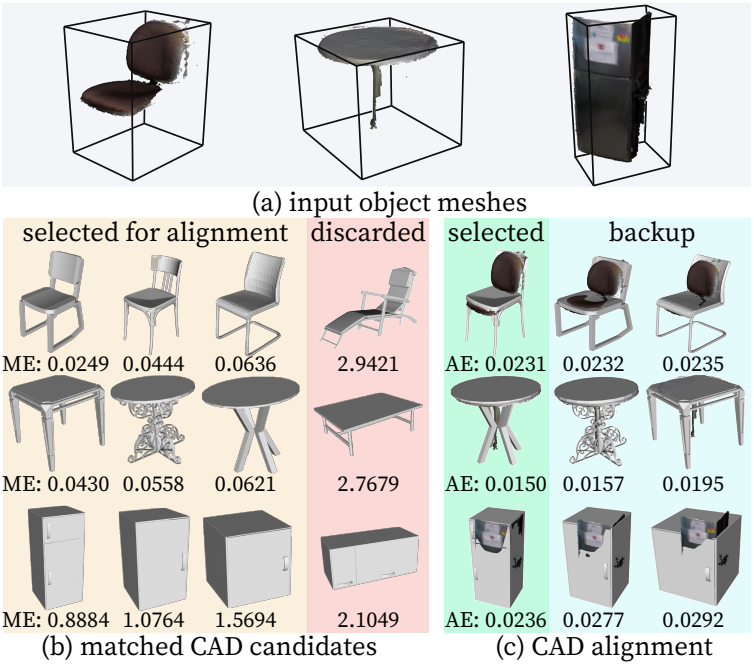


Figure 2.5: **Examples of matching and aligning CAD candidates** to (a) input object meshes. (b) After selecting the CAD candidates with smallest MEs, (c) a CAD alignment process selects the best CAD model with a proper transformation based on Alignment Error (AE).

2.4.3 Optimization-based CAD Alignment

The overarching goal of this step to find an accurate transformation (instead of 24 discretized orientations in the previous step) that aligns a given CAD candidate y to the original object entity x, achieved by estimating a homogeneous transformation matrix between x and y:

$$T = \begin{bmatrix} \alpha R & \boldsymbol{p} \\ \mathbf{0}^T & 1 \end{bmatrix}, \; s.t. \; \min_{T} \mathcal{J}(x, T \circ y), \tag{2.11}$$

where $\circ$ denotes the transformation of a CAD candidate y, $\mathcal{J}$ is an alignment error function, α is a scaling factor, $R = Rot(\boldsymbol{z}, \theta)$ is a rotation matrix that only considers the yaw angle under the gravity-aligned assumption, and $\boldsymbol{p}$ is a translation. This translation is subject to the following constraint: $p^g = -d^s + \alpha \cdot s^g_y/2$, as the aligned CAD candidate is supported by a supporting plane $\boldsymbol{\pi}^s = [\boldsymbol{n}^{sT}_{\cdot}, d^s_{\cdot}]$.

The objective function $\mathcal{J}$ can be written in a least squares form and minimized by the Levenberg – Marquardt [Mor78] method:

$$\mathcal{J} = \boldsymbol{e}_b^T \Sigma_b \boldsymbol{e}_b + \boldsymbol{e}_p^T \Sigma_p \boldsymbol{e}_p, \tag{2.12}$$

where $\boldsymbol{e}_b$ is the 3D bounding box error, $\boldsymbol{e}_p$ the plane alignment error, and Σ_b, Σ_p the error covariance matrices of the error terms. Specifically: (i) $\boldsymbol{e}_b$ aligns the height of the two 3D bounding boxes while constraining the ground-aligned rectangle of the transformed B_y inside that of B_x:

$$\boldsymbol{e}_b = [\mathrm{A}(T \circ y) - \mathrm{A}(x \cap T \circ y), \alpha \cdot \boldsymbol{s}_y^g - \boldsymbol{s}_x^g]^T, \tag{2.13}$$

and (ii) $\boldsymbol{e}_p$ aligns all the matched feature planes as:

$$\begin{aligned} \boldsymbol{e}_p &= [\Delta\boldsymbol{\pi}_1, ..., \Delta\boldsymbol{\pi}_{|\Pi_x|}]^T, \\ \Delta\boldsymbol{\pi}_i &= [-d(\boldsymbol{\pi}_i) + d(T^{-T} \cdot f_\Pi(\boldsymbol{\pi_i})), \\ &\qquad 1 - \boldsymbol{n}(\boldsymbol{\pi}_i)^T \cdot \boldsymbol{n}(T^{-T} \cdot f_\Pi(\boldsymbol{\pi}_i))], \end{aligned} \tag{2.14}$$

where some of the notations are detailed in Section 2.2.

To evaluate how well an aligned CAD candidate fits the object mesh, we compute an AE defined as the root mean square distance between the object mesh vertices and the closest points on aligned CAD candidate; Fig. 2.5c shows both qualitative and quantitative results. The CAD candidate with the smallest AE will be selected, whereas others are potential substitutions if the selected CADs violate physical constraints, detailed next.

2.4.4 Global Physical Violation Check

Given a shortlist of matched and aligned CAD candidates, we propose a global physical violation check to finalize the CAD replacement and generate a physically plausible cg'. We first validate supporting relations and object-layout proximal relations for CAD candidates of each object. Specifically, for an object node v_p and its segmented object entity x, we discard an aligned CAD candidate y if it fails to satisfy Eq. (2.2) with any supporting child v_c of v_p. We also discard aligned CAD candidates that violate the proximal constraints with layout entities.

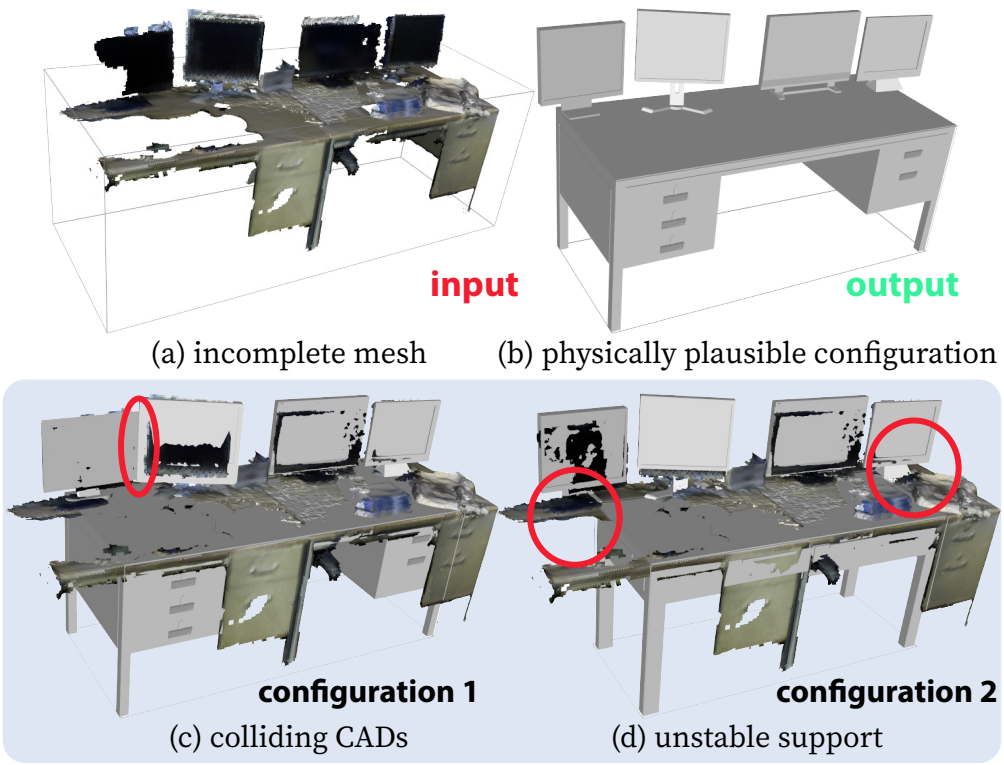


Figure 2.6: **An example of global physical violation check** that prunes invalid configurations such as (c) collision and (d) unstable support, and produces (b) a physically plausible configuration.

After early discard of invalid CAD candidates, we check the inter-object proximal constraints and jointly select CAD candidates for each object entity. We address this by formulating a constraint satisfaction problem; starting with a CAD candidate with the minimum AE for each segmented object, we adopt the min-conflict algorithm [MJP92] to obtain a global solution of CAD replacement. Finally, as the CAD alignment step cannot guarantee the precise alignment of supporting planes, we adjust the position of CAD models so that Eq. (2.1) is strictly satisfied for each supporting relation. Then we obtain a finalized cg' with CAD models.

Fig. 2.6 illustrates a typical example, where specific configurations of CAD replacements lead to unstable support or colliding geometry. Then the abovementioned global physical violation check prunes invalid configurations and outputs a physically plausible one.

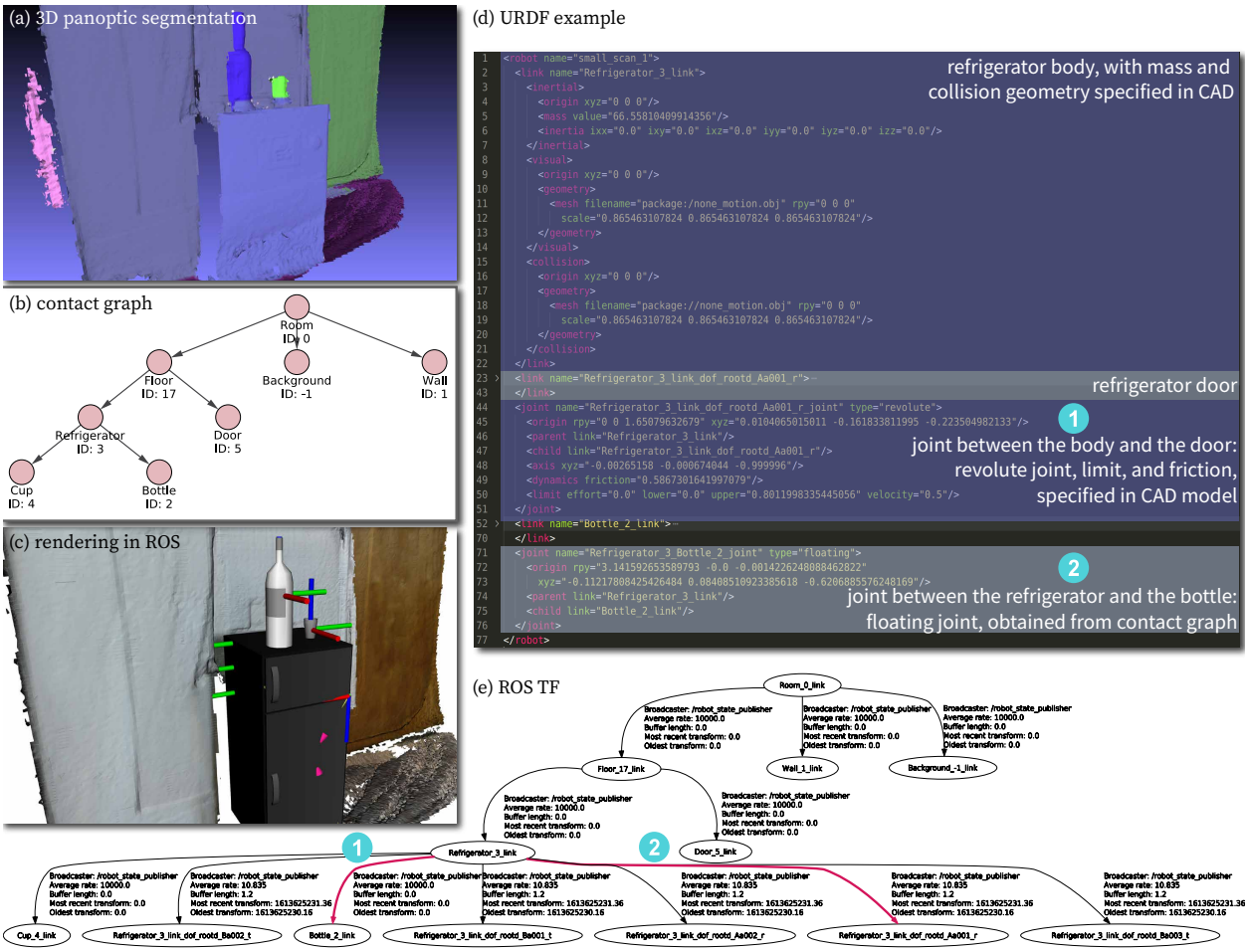


Figure 2.7: **Convert a contact graph cg′ to a kinematic tree.** (a) Given the 3D panoptic segmentation produced by our mapping module, (b) a contact graph is built and converted to (d) Unified Robot Description Format (URDF) with CAD models, which can be seamlessly (c) imported to and visualized in ROS Rviz; (e) the corresponding ROS TF describes the world states to robots.

2.4.5 Kinematic Tree Conversion

The finalized cg' can be readily converted into a kinematic tree to support various robot planning tasks. In this work, we develop an interface to generate a kinematic tree in the form of Unified Robot Description Format (URDF), which is commonly used in the robotics community.

A kinematic tree contains rigid bodies (links) as nodes, and joints connecting two bodies as

edges. Each node in the kinematic tree can be created from either a scene root node, a layout node, a rigid object node, or a rigid part of an articulated object node in cg'. We preserve the joints within articulated CAD models in the kinematic tree, but alter the supporting edges in cg' to either fixed joints (no translation or rotation allowed) or floating joints (allow 3D translation and 3D rotation unless is constrained by collision) based on the semantics of the scene entity pairs. For example, a cup is connected to a table using a floating joint as a robot can freely manipulate it, and a table is linked to the floor via a fixed joint as it cannot be moved.

We show a detailed example of the kinematic tree conversion process in Fig. 2.7. Based on the 3D panoptic segmentation and the contact graph, our interface generates a kinematic tree in URDF, which can be further visualized as ROS TF and rendered in ROS Rviz. In this example, the fridge is connected to the floor via a fixed joint, and the bottle to the fridge via a floating joint. A revolute joint connects the fridge body and the fridge door as specified by the CAD model.

2.5 Experiments and Results

2.5.1 Dataset and Implementation

We evaluate our system primarily on the SceneNN dataset [HPN16]; it contains RGB-D sequences of various room-size indoor scenes and ground-truth scene meshes annotated with instance-level segmentation. We pick 20 test sequences/scenes that contain diverse object categories to quantitative evaluate the robust panoptic mapping module and demonstrate the interactive scene reconstruction. For baselines that require training on 3D segmentation data, we roughly follow the train/test split in [HTY18] while using the test set we pick.

In our work, we choose the baseline panoptic segmentation model in Detectron2 [WKM19], pre-trained on the COCO panoptic class [LMB14] for segmentation on RGB. We use [FNF18] as the baseline geometric segmentation method for depth images. Of note, our system is designed in a modularized manner so that it is flexible enough to incorporate more powerful models when

available. For instance, the segmentation module is designed as a server-side service that will be requested by a client in the perception system when a new image frame arrives and produce a list of segmented masks with labels in the response. Any segmentation methods being wrapped as a service following this protocol could be connected to our system.

2.5.2 Robust Panoptic Mapping

We evaluate our robust panoptic mapping module on three aspects: (i) 3D panoptic mapping quality, (ii) 3D object instance segmentation, and (iii) oriented 3D bounding box estimation. The first aspect focuses on how well the system reconstructs the scene and segments the objects and layouts within, whereas the latter two emphasize individual objects. Such a protocol design provides a holistic evaluation of the fundamental component of the proposed system: The accuracy of object segmentation and bounding box estimation are crucial for the overall quality of scene reconstruction when matching and aligning CAD models. An ablation study (noted as "w/o joint fusion") is also conducted, where we disable our modifications of jointly processing semantic and instance labels in data fusion, *i.e*. the procedure described in Sections 2.3.2 and 2.3.3. This study will not only better demonstrate how much the introduced modifications influence the overall mapping performance, but also verify the effectiveness of the per-frame segmentation and fusion technique by comparing the ablated results with those from baselines.

For each sequence used in the experiment, our mapping module processes incoming RGB-D frames with ground-truth camera poses provided by the dataset. We consider 10 semantic classes including 2 *stuff* classes (wall and floor) and 8 most common *thing* classes (bed, table, chair, monitor, sofa, bag, cabinet, and fridge) for evaluation.

3D Panoptic Mapping This experiment evaluates the overall segmentation performance for panoptic mapping, following the criteria defined in [KHG19] and [NSI19]:

$$PQ = \underbrace{\frac{\sum_{(p,g)\in TP} \text{IoU}(p,g)}{|TP|}}_{\text{SQ}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{RQ}}, \tag{2.15}$$

where the Segmentation Quality (SQ) is the averaged Intersection over Union (IoU) of $p$redicted and $g$round-truth panoptic masks on all matched predictions in the same class, and the Recognition Quality (RQ) is the F_1 score [MFM04] of object recognition for the aforementioned 10 semantic classes. Panoptic Quality (PQ) is simply the product of SQ and RQ, which better reflects the overall segmentation results.

We compare our panoptic mapping module with the Voxblox++ [GFN19]. Table 2.1 (white columns) shows their corresponding PQ, RQ, and SQ of 7 individual SceneNN sequences, averaged on 10 classes. Table 2.2 further tabulates per-class panoptic segmentation results of all 20 sequences. Of note, we compute PQ, RQ, and SQ in category-level for each semantic class (Table 2.2), and average the PQ, RQ, and SQ of all classes to obtain those values in scene-level (Table 2.1).

Overall, our panoptic mapping module significantly outperforms the baseline as indicated by higher PQ for individual sequences and most of the semantic classes. Without applying joint fusion, our system still performs better than the baseline Voxblox++, showing the efficacy of our per-frame segmentation. But it is not as good as our full module, which further demonstrates that our proposed strategies positively contribute to objects and layouts recognition (higher RQ value indicates higher accuracy) and segmenting them well (higher SQ value). The extra performance gain our modifications bring is very crucial for the subsequent processes.

3D Instance Segmentation We also evaluate the performance of 3D instance segmentation on 8 *thing* classes using the mAP@0.5 metric, *i.e.*, the Maximum a Posteriori (MAP) computed using an Intersection over Union (IoU) with a threshold of 0.5. The evaluation is two-fold. First, we report the class-averaged results in the progressive mapping manner on 7 individual sequences

Table 2.1: **Quantitative class-averaged results of 3D panoptic segmentation and 3D instance segmentation on individual sequences in the SceneNN dataset [HPN16].** Note that ProgressFusion [PHN19] accounts for more classes than the other two methods. All values are in percentage.

	Ours				Voxblox++ [GFN19]				ProgressFusion [PHN19]
	Panoptic			Instance	Panoptic			Instance	Instance
ID	PQ	SQ	RQ	mAP	PQ	SQ	RQ	mAP	mAP
011	**45.5**	60.4	50.0	58.3	34.3	64.3	40.0	**80.8**	52.1
030	**50.4**	55.6	64.5	**58.3**	23.4	34.7	26	33.5	56.8
061	**43.0**	52.0	46.3	33.6	25.7	53.1	32.2	38.6	**59.1**
078	**54.7**	54.7	62.5	**50.0**	26.3	52.5	31.7	43.9	34.9
086	**27.3**	39.6	34.6	**40.8**	19.4	32.9	25.2	37.6	35.0
096	**12.5**	21.4	14.6	23.0	7.3	11.9	8.3	14.6	**26.5**
223	**49.5**	60.2	63.3	**60.0**	21.7	40.2	26.7	34.1	40.9

Table 2.2: **Per-class 3D panoptic segmentation results in the SceneNN dataset.** All values are in percentage.

		all	stuff	thing	wall	floor	bed	table	chair	monitor	sofa	bag	cabinet	fridge
Voxblox++ [GFN19]	PQ	24.5	10.9	27.9	4.0	17.8	**18.0**	14.4	35.5	48.5	**46.0**	**24.0**	7.2	29.5
	SQ	77.6	73.7	78.6	69.3	78.0	72.0	71.3	77.0	81.4	82.8	84.0	86.0	73.9
	RQ	31.2	14.3	35.4	5.7	22.9	25.0	20.3	46.0	59.6	55.6	28.6	8.3	40.0
Ours (w/o joint fusion)	PQ	27.8	12.6	31.6	5.6	19.5	8.7	26.7	31.7	48.8	45.7	16.1	**21.9**	**53.4**
	SQ	77.5	71.8	78.9	64	79.6	65.9	73.8	76	89	82.2	72.6	78.5	93.4
	RQ	34.2	16.6	38.6	8.7	24.5	13.3	36.1	41.8	54.9	55.6	22.2	27.9	57.1
Ours	PQ	**35.4**	**44.2**	**33.2**	**25.2**	**63.1**	11.5	**27.4**	**40.1**	**65.7**	34.3	17.4	20.1	48.7
	SQ	80.5	79.3	80.9	73.5	85.0	77.6	76.1	79.1	88.8	80.0	78.3	81.7	85.2
	RQ	43.1	54.3	40.3	34.3	74.3	14.8	36.0	50.6	73.9	42.9	22.2	24.6	57.2

Table 2.3: **Per-class 3D instance segmentation results on the SceneNN dataset.** The numbers in bold and numbers in underscore indicate the best and the second best results, respectively. All values are in percentage.

	Input Format	bed	table	chair	monitor	sofa	bag	cabinet	fridge
MT-PNet [PNH19]	Full point cloud	0.0	12.5	42.8	26.5	0.0	0.0	0.0	0.0
MLS-CRF [PNH19]	Full point cloud	0.0	27.3	50.9	38.6	0.0	0.0	0.0	0.0
OccuSeg [HZX20]	Full point cloud	**66.7**	**50.0**	**91.3**	**76.9**	50.0	-	5.7	-
Voxblox++ [GFN19]	RGB-D stream	39.4	22.3	55.6	63.6	72.4	**56.4**	8.5	51.6
Ours (w/o joint fusion)	RGB-D stream	17.4	40.7	51.3	48.1	**82.8**	53.2	35.4	**94.5**
Ours	RGB-D stream	27.5	46.6	65.3	69.4	64.3	53.2	**43.9**	**94.5**

Table 2.4: **Per-class oriented 3D bounding box estimation results (mAP@0.5) on the SceneNN dataset [HPN16].** All values are in percentage.

	all	bed	table	chair	monitor	sofa	bag	cabinet	fridge
MT-PNet [PNH19]	10.4	25.8	12.8	19.3	25.0	0.0	0.0	0.0	0.0
MLS-CRF [PNH19]	5.7	0.0	12.6	33.0	0.0	0.0	0.0	0.0	0.0
Voxblox++ [GFN19]	24.1	**39.4**	19.5	31.8	37.0	47.9	0.0	4.0	13.4
Ours (w/o joint fusion)	28.5	17.4	21.4	36.6	29.4	55.8	**53.2**	14.1	0
Ours	45.3	27.5	54.9	44.6	**42.5**	53.7	**53.2**	**29.8**	**56.4**
Ours (refined)	**47.2**	22.9	**68.2**	**49.2**	38.7	**59.1**	**53.2**	**29.8**	**56.4**

compared with Voxblox++ [GFN19] and ProgressFusion [PHN19], another online semantic mapping framework; see the grey columns in Table 2.1. Our approach performs better than Voxblox++ on almost all the sequences. Note that ProgressFusion accounts for all NYUDv2 [SHK12] classes available in the dataset, and we evaluate the performance only on the 8 *thing* classes for our method and Voxblox++. While it's possible to re-train our panoptic segmentation module to incorporate

more classes, we believe the current experiment is sufficient to demonstrate the advantage of our panoptic mapping module without defeating its purpose of leveraging pre-trained perception models.

Second, in Table 2.3, we study the per-class mAP@0.5 of our approach compared with the baseline Voxblox++ [GFN19] and two learning-based works [PNH19, HZX20] that directly segment 3D instances from the full point cloud of scenes instead of continual RGB-D data stream. As the input formats are different, the results are not directly comparable. They nevertheless provide a better sense about how well our approach performs. We re-train [PNH19] and report the results of its two variants on our test set, and adopt the results reported in [HZX20]. Overall, our method performs significantly better than Voxblox++ in most classes, and our variant without joint fusion can still slightly outperform Voxblox++. OccuSeg appears to perform the best for object classes that are less likely to be severely occluded in the dataset, while our approach poses a unique advantage of handling partially-visibly objects such as cabinets and fridges usually attached to a wall.

Oriented 3D Bounding Box Estimation We further evaluate the accuracy of oriented (gravity-aligned) 3D bounding boxes of object instances, which serve as essential geometric cues for physical reasoning and CAD replacement. Similarly, the mAP@0.5 metric is adopted to evaluate the oriented 3D bounding box estimation on the 8 *thing* classes. Table 2.4 tabulates results using the baseline method [GFN19], two variants described in [PNH19], our approach, and our approach with supporting-based refinement (detailed in Section 2.2.2). Note that since there is no native support for evaluating oriented 3D bounding boxes in [PNH19], we re-train the models on the SceneNN dataset for this experiment. The results indicates that our approach predicts their oriented 3D bounding boxes accurately for most object classes compared with the baselines. The refinement process further improves the performance by completing the partially-observed object boxes. Looking at the two variants in [PNH19], while MLS-CRF introduces an extra post-processing step using a Conditional Random Field (CRF) on top of the MT-PNet, its 3D bounding box estimation accuracy drops as extra points from the background are merged into the foreground objects

Table 2.5: **Graph Editing Distance (GED) of four scenes between annotated cg_{gt} and inferred contact graph from our panoptic mapping results cg_{ours} (Fig. 2.9b) and from ground-truth maps cg_{map} (Fig. 2.9a).** Note that editing a wrong support will need two operations, removing an edge and adding an edge, resulting a graph distance of 2.

Scene	Total nodes		Total distance		Wrong support		Missing detection		Wrong detection	
	cg_{gt}	$v.s.$	cg_{ours}	cg_{map}	cg_{ours}	cg_{map}	cg_{ours}	cg_{map}	cg_{ours}	cg_{map}
225	20		12	4	1	2	5	0	5	0
231	29		9	4	0	2	2	0	7	0
249	11		7	0	3	0	1	0	0	0
322	17		5	2	1	1	2	0	1	0

in CRF regularization. An interesting disparity between [PNH19]'s instance segmentation results (Table 2.3) and its bounding box estimation (Table 2.4) appears—having a zero-score in one place and turning to positive in another. This is because a subtle change in segmenting instances may lead to a large error in estimated bounding boxes.

In summary, the above three quantitative evaluations demonstrate that our robust panoptic mapping module well suited for (i) recognizing and segmenting scene entities progressively during mapping and (ii) estimating objects' 3D oriented bounding boxes in complex and clustered real indoor environments. The former capability is essential for selecting a proper CAD model to replace a segmented object, and the latter determines the size and scale of that CAD. The ablation study highlights the performance gain introduced by our data fusion procedure, demonstrating the success of jointly dealing with semantic and instance predictions during mapping.

2.5.3 Inferred Contact Graph

Having extracted object and layout meshes from the volumetric panoptic map, a contact graph cg can be built based on inferred supporting relations before using it to bridge the actual scene to a

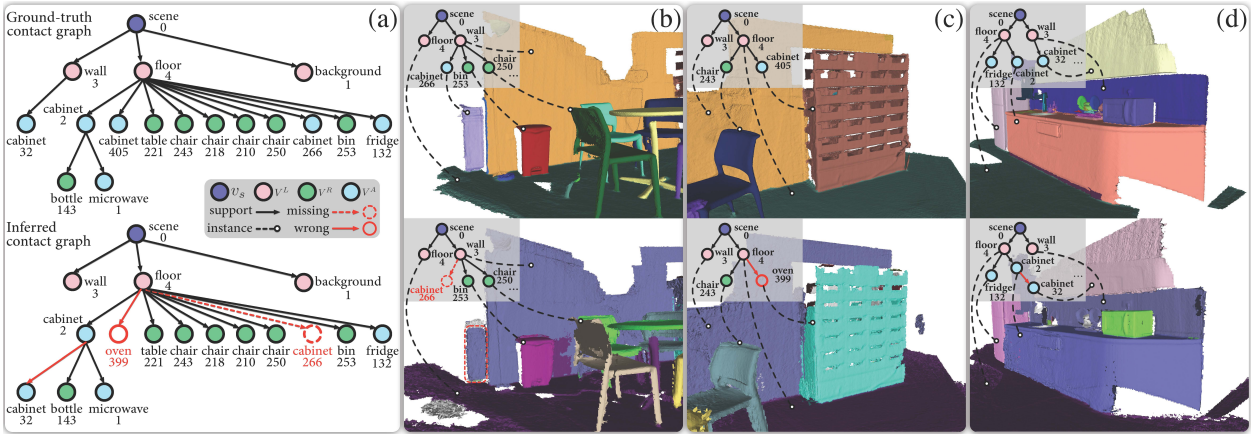


Figure 2.8: **Comparison between ground-truth and inferred contact graph**. (a) The annotated cg_{gt} $v.s.$ the cg_{ours} inferred from our panoptic mapping results for scene 322. (b)(c)(d) highlight a missing detection (cabinet 266 is not detected), a wrong detection (cabinet 405 is detected as oven 399), and a wrong support (cabinet 32 is supported by wall), respectively.

virtual one. It is worthwhile to evaluate the structure of an inferred cg as it collectively reveals the performance of object recognition, supporting relation estimation, and overall results. To conduct this evaluation, we annotate the contact graphs of four scenes in the SceneNN dataset [HPN16] based on their ground-truth segmentation shown in Fig. 2.9a. Then, a Graph Editing Distance (GED) [ZS89] metric is applied to evaluate the distance between an annotated contact graph and an inferred graph from a segmented map. Specifically, GED measures the dissimilarity of two graph by how many graph editing operations (we consider five operations, *i.e.*, insertion, removal of a node or an edge, and substitution of a node ID) are needed to convert one graph to the other.

The results are reported in Table 2.5, where we compare the GED between (grey columns) the annotated contact graph cg_{gt} and that inferred from our mapping results cg_{ours}, and between (white columns) cg_{gt} and that inferred from ground-truth segmentation map cg_{map}. The Total nodes column indicates the size of cg_{gt}, *i.e.* the number of scene entities a scene has. The Total distance column shows the total editing operations required to covert cg_{ours} or cg_{map} to cg_{gt}, indicating the overall quality of the inferred cg. A qualitative illustration between two graphs is also shown in Fig. 2.8a. Moreover, the GED can be broken down to three types of errors appeared in an inferred

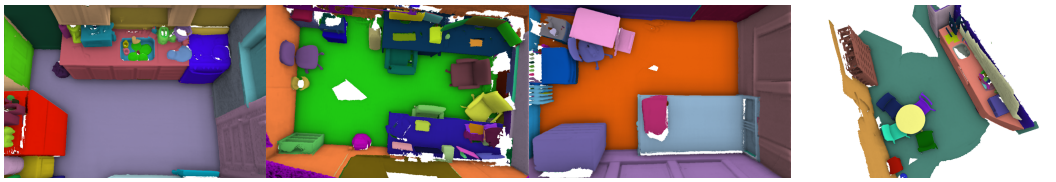
graph: (i) Wrong support (or wrong edge): a supporting relation is not assigned correctly, *i.e*. the parent node of an entity should be another; (ii) Missing detection (or missed node): an entity is not detected or segmented and thus not included in the graph; (iii) Wrong detection (or extra node): an entity that is not supposed to appear in the graph, and the reasons for having extra nodes could be having a wrong semantic label, one entity is segmented as multiple ones, or both. Fig. 2.8bcd depict some examples of error in scene 322.

In Table 2.5, we observe that our system mainly suffers from the clustered scene 225 and scene 231 with lots of small objects, indicated by the high costs of Missing and Wrong detection. On the other hand, the relatively low cost caused by Wrong support indicates that our criteria of determining supporting relations is effective.

2.5.4 Interactive Scene Reconstruction

Fig. 2.9 showcases the qualitative results for reconstructing functionally equivalent and interactive scenes. Given a volumetric panoptic map (Fig. 2.9b) and a constructed contact graph, our system reconstructs a high-quality, functionally equivalent, interactive scene by replacing incomplete meshes with CAD models and perform physical reasoning on the contact graph, as shown in Fig. 2.9c. Nevertheless, we find that our system performs poorly or fails under two circumstances: (i) The incomplete object mesh has misguided or no feature planes, resulting in the misalignment of the CAD model; (ii) The object is not supported by its bottom face (*e.g*., cabinets on the wall), resulting in the incorrectly reconstructed scene due to the wrong estimate of the supporting relations. Section 2.6 provides a more in-depth discussion of the system limitations.

By converting the scene contact graph into a kinematic tree in URDF, we are able to seamlessly import the reconstructed functionally equivalent and interactive scene into various existing simulators. Practically, we also specify physical proprieties (such as link mass, collision geometry, joint friction) in URDF to facilitate more sophisticated simulations. We demonstrate the usage of our reconstructed interactive scenes with several examples: (i) Fig. 2.9d shows the reconstructed

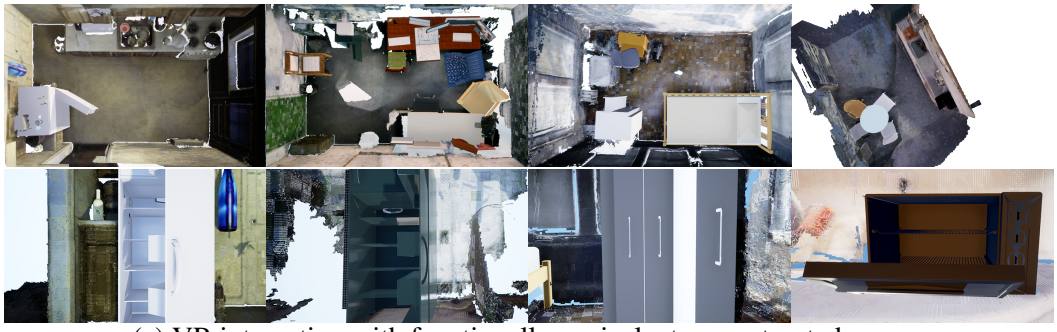


(a) Ground-truth segmentation for scene 225, 231, 249, and 322 [HPN16]

(b) Segmentation results produced by the proposed panoptic mapping

(c) Scene reconstruction with functional and actionable CAD objects

(d) Robot interaction with functionally equivalent reconstructed scenes



(e) VR interaction with functionally equivalent reconstructed scenes

Figure 2.9: **Qualitative results of four reconstructed scenes with actionable CAD models.** Both robots and human users can virtually enter the reconstructed interactive scenes for Task and Motion Planning (TAMP) and VR applications.

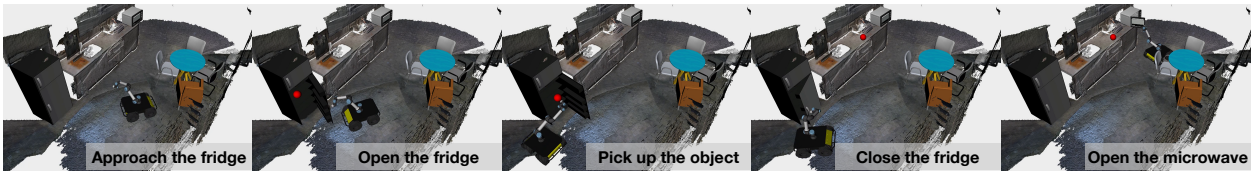


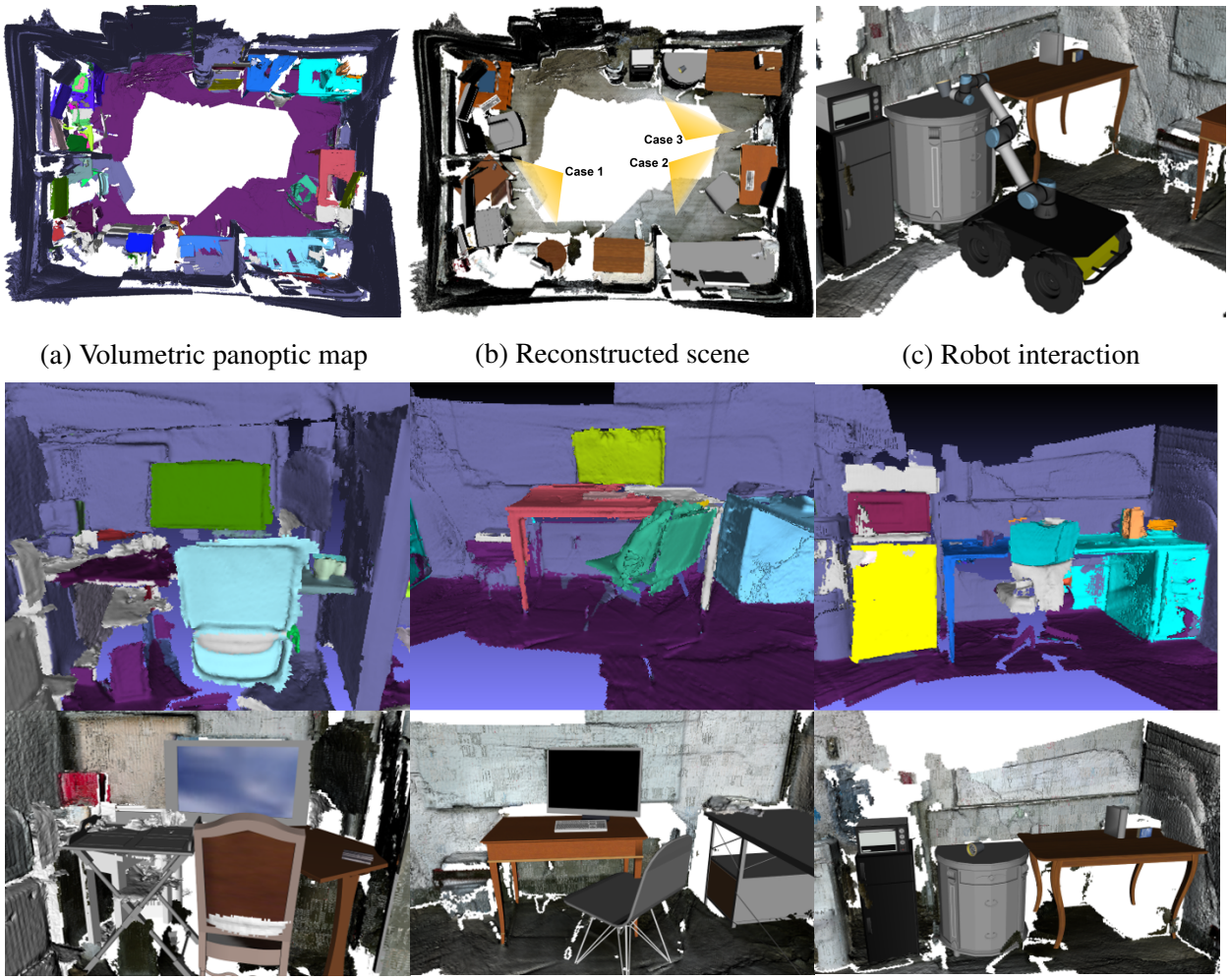
Figure 2.10: **Robot executing a mobile manipulation task with multiple steps**: microwaving an item (indicated by the red ball) by first retrieving it from the fridge.

scenes in the ROS environment, which subsequently connects the reconstructed scenes and robot Task and Motion Planning (TAMP). Detailed planning schemes and implementations could be found in the authors' parallel work [JZW21b, JZJ21]. (ii) Fig. 2.9e demonstrates that the reconstructed scenes can be loaded into the VR environment [XLZ19] for interactions with both virtual agents and human users, which opens a new avenue for future studies. (iii) Fig. 2.10 presents keyframes of a robot executing a long-horizon mobile manipulation task that involves interactions with articulated objects.

2.5.5 Reconstruction of Physical Scenes

To further evaluate our system under a real-world setting, we conduct experiments to reconstruct physical scenes using a handheld Kinect v2 sensor. We obtain accurate camera poses with a state-of-the-art feature-based SLAM system [MT17] based on RGB-D streams. The resulting 3D volumetric panoptic map, reconstructed functionally equivalent and interactive scene, and an example of robot interaction are shown in Figs. 2.11a to 2.11c, respectively. This result reveals a huge potential of applying the proposed system to facilitate robot task execution in the physical world.

We further analyze scene reconstruction results using three typical cases that highlight the advantages and failure conditions. In case 1 (Fig. 2.11d), the table is occluded by the chair and thus is identified as two instances floating in the air. These two tables are determined as floor-supported, and their 3D bounding boxes are further refined on the basis of the supporting relations. The system eventually outputs two separate tables in the reconstructed interactive scene, where their poses aligned with the oriented 3D bounding boxes of the partial meshes. Case 2 (Fig. 2.11e) shows



(a) Volumetric panoptic map (b) Reconstructed scene (c) Robot interaction

(d) Case 1 (e) Case 2 (f) Case 3

Figure 2.11: **Reconstructing a physical scene with a handheld RGB-D sensor**. (a) The panoptic segmentation and the overall mapping. (b) The reconstructed scene with CAD models replacing the segmented objects, which supports (c) a robot to simulate its Task and Motion Planning (TAMP). (d–f) Qualitative results of segmentation and reconstruction. Our system recognizes most of the objects and properly replaces them with CAD models that are similar to those objects in the physical scene; see Case 2 and 3. A common problem is due to occlusion, which causes inaccurate detection, *e.g.*, one desk is recognized as two as it is occluded by the chair; see case 1 and 3.

an example of a better reconstructed workspace. Given the incompletely segmented table and chair point cloud, our system can correctly estimate the supporting relations and their orientations, replace each mesh with a similar CAD model, and finally produce a functionally equivalent and physically plausible workspace, although the dimension of the table is not ideal as part of the point cloud behind the chair is not detected and segmented correctly. Case 3 (Fig. 2.11f) provides a more challenging example. The fridge and microwave are segmented and replaced by articulated CAD models, whereas the chair is not successfully detected and is removed from the reconstructed scene. Similar to case 1, the table is identified and replaced with two instances. To avoid mesh penetration, the proximal constraints incorporated by the cg helps the CAD replacement process to select a rounded table on the left side, but it is not a satisfactory replacement due to the large discrepancy in shapes.

2.6 Discussions

2.6.1 Scene Functionality

Most computer vision tasks focus on devising new methodologies and representations that are beneficial within the scope of computer vision. However, this work seeks to address a new task of building a representational system with the emphasis of facilitating robot activities. The core of the system is to represent the scene *functionality*, one of the key common senses governing our understanding of a scene [ZGF20]. This goal is achieved by associating high-level cues from object semantics (*e.g.*, whether they can be moved, opened, or can support other interactions) and low-level cues (*i.e.*, replacing the object meshes with CAD models, whose underlying kinematic indicate how exactly they interact). Additional object attributes, affordance, or task-dependent information can be annotated to CAD models to depict the scenes more comprehensively. A subsequent, interesting open question is how to quantify the divergence between the actual scene and the reconstructed one with CAD replacements.

2.6.2 Scene Representation

The contact graph cg produced by the proposed system is a holistic, but approximate scene representation. By itself is indeed insufficient for robot task executions where more precious local scene representations are needed. Although the cg does not seem directly beneficial, its importance is two-fold when considering a robot designed to operate over a long period of time. Firstly, the representation maintains a global belief of the scene, helps a robot to anticipate the effects of (sequence of) actions, and incorporates the actual action effects back to the cg. This is essential for the robot to forward search for a task plan over a long horizon [Kae20]. Secondly, given the variety of tasks a robot may anticipate, our cg can serve as a carrier for those necessary local representations that can be annotated, trained beforehand or build online with proper perception modules. Otherwise, different task-driven representation are standalone, lacking proper organizations.

2.6.3 Task and Motion Planning (TAMP)

Existing TAMP frameworks are oftentimes too brittle to handle a large variety of the environment for interactions. [KL11] and [SFR14] propose new TAMP frameworks, making planning long-horizon manipulation tasks possible. Still, the framework focuses on pick-and-place tasks with carefully defined environmental constraints, making it difficult for complex indoor manipulation tasks. [GPL20] devise a framework for a complex problem, which requires interaction with articulated objects. Similarly, this work is still limited to carefully designed environments with limited variety in the setup. A key factor to this problem is the lack of simulation environments that support various interactive actions (*e.g.*, door opening, object picking) and semantic relations among objects. Crucially, it could be time-consuming to generate these environments manually. In comparison, our framework can automatically generate interactive environments from real sensory data of challenging physical world in the wild and demonstrate a certain capability to support more complex TAMP study in the future.

2.6.4 Embodied AI

Embodied AI researches focus on learning a policy, mostly in simulations, that can ultimately be applied to real-world applications. Therefore, a significant amount of work is to develop simulation platforms to support learning. Our perspective echoes the motivation of task-oriented vision—designing a proper vision system that better suits a given task [IH92]. Specifically, our work allows the agent to acquire a policy specific to the given environment for the given task by capturing and representing the *actionable* information in the environment from the agent's view. Thus, our work goes beyond panoptic segmentation and 3D reconstruction.

2.6.5 Supporting Relations

Inferred supporting relations define the structure of contact graph. While this work mainly concerns about stable support, *i.e*. those satisfying Eq. (2.3), there are several other supporting configurations, such as an object is hanging on wall and supporting from behind, is supported by two adjacent tables, is placed on floor and tilted against another object *etc*. These types of supports are not explicitly modeled and may not be well handled. Our system can nevertheless reveal their supporting relations in part. For instances, the blue bottle in Fig. 2.1c is regarded as supported by the wall because no valid supporting parent is identified but it is very close to the wall. Whereas in Fig. 2.8d, the upper cabinet that is supported by the wall (and possible the ceiling as well) is wrongly considered as supported by the lower cabinet. In other cases where an object is supported by multiple entities simultaneously, only one entity would be identify as a supporting parent based on overlapping area defined in Eq. (2.2). For a tilted object on floor, only the floor would be identified as the supporting object. Hanging objects that are supported from above, are not handled in the present work either. Apparently, our strategy cannot fully address the above less common supporting relations reliably at all time, but more specific spatial relations can be modeled and incorporated into the contact graph representation as well to extend the system's capability.

2.6.6 Other Limitations

The system's performance heavily relies on 3D panoptic segmentation of scene entities and the CAD replacement of object meshes. Currently, our robust panoptic mapping module utilizes open-sourced software to generate panoptic segmentation on RGB frames. While its development is beyond this work's scope, new models and methods are emerging in the fast-paced community, and our system is designed to easily incorporate newer methods to improve the mapping performance further and support subsequent processes by reducing error propagated in each stage.

Our CAD replacement algorithm matches and aligns CAD models to incomplete meshes based on simple geometric features, *i.e.*, 3D bounding boxes and surface planes, which are potentially fragile when the meshes are noisy and incomplete. In the future, we may integrate deep learning-based methods [ADN19, PTL18] for more robust and accurate CAD replacement.

The articulated CAD models are unlikely to match the structure of real objects exactly. One potential solution is to detect and segment object parts and estimate the kinematics to assemble fine-grained CAD models. The PartNet dataset [MZC19] provides an initial direction to start with.

There are various actionable information and many other information an object should afford for a robot to sufficiently interact with it depending on different task specifications, while this work only studies a few, *e.g.* inferred supporting relations and annotated kinematics information. One central question remains unanswered is how to balance manual efforts and algorithmic efforts so that an intelligent robot can better excel in ever-changing environment.

2.7 Conclusion

This chapter presents a contact graph representation to bridge raw perception and robot TAMP, as well as a new task of reconstructing functionally equivalent and interactive scenes to simulate robot autonomy. We develop a perception system to demonstrates this new perspective. Contrasting to the classic view of scene reconstruction that focuses on the geo-information, our system captures

semantics and associated actionable information in scene entities by (i) a novel panoptic mapping module that reconstructs individuals objects and layouts, (ii) a geometric and physical reasoning module to replace the incomplete objects meshes with part-based interactive CAD models, and (iii) a contact graph representation that facilitates physically plausible scene reconstruction, and reflects action opportunities and action outcomes in terms of kinematic information. In experiments, we quantitatively demonstrate that our system can produce high-quality panoptic segmentation, and qualitatively showcase various reconstructed scenes with functional CAD model replacements that support fine-grained interactions in ROS and VR environments.

Part II

Planning: Closed-loop Planning with Language Models and Environment Feedback

CHAPTER 3

Closed-Loop Mobile Manipulation with Vision Language Model

Autonomous robot navigation and manipulation in open environments require reasoning and replanning with closed-loop feedback. In this chapter, we present COME-robot, the first closed-loop framework utilizing the GPT-4V vision-language foundation model for open-ended reasoning and adaptive planning in real-world scenarios. We meticulously construct a library of action primitives for robot exploration, navigation, and manipulation, serving as callable execution modules for GPT-4V in task planning. On top of these modules, GPT-4V serves as the brain that can accomplish multimodal reasoning, generate action policy with code, verify the task progress, and provide feedback for replanning. Such design enables COME-robot to (i) actively perceive the environments, (ii) perform situated reasoning, and (iii) recover from failures. The materials in this chapter have been published as a preprint [ZZH24].

3.1 Introduction

Developing autonomous robots capable of navigating and manipulating objects in unknown, real-world environments remains a long-standing challenge [NO00, PJT02, GCB23, LOP24]. To effectively operate in long-horizon tasks, robotic systems must meet several critical requirements: (i) the ability to actively gather multi-modal environment feedback using on-board sensors, (ii) the capability to robustly execute primitive actions such as exploration, navigation, and object manipulation, and (iii) the capacity for reasoning to interpret feedback from both environment perception and action execution, and adjust action plans accordingly. Such a closed-loop system plays a vital role in enabling robots to address issues of partial observability and recover from perception and

execution failures inherent in complex, unstructured real-world environments. In the literature, traditional closed-loop replanning frameworks [GPL20, CFK22] typically depend on predefined symbolic logic for reasoning and replanning. These frameworks rely on specialized perception models to process raw observations (e.g., images or point clouds) into a form suitable for symbolic reasoning. Consequently, they encounter difficulties when confronted with open-ended, real-world scenarios that demand adaptation to unforeseen circumstances.

Recent advances in large foundation models offer promising avenues for augmenting robot intelligence [FTT23]. Large language models (LLMs) have demonstrated remarkable abilities to elucidate failures, propose corrections, and devise new plans based on textual environmental feedback [HXX23, YZY22, LBS23]. Moreover, they can generate computer programs (*i.e.*, code) with modularized perception and execution API function calls to interface robots' task execution [LHX23, SBM23]. Vision-language models (VLMs) further extend these capabilities by enabling open-vocabulary (*i.e.*, free-form instructions or descriptions of scenes and objects) understanding of raw visual observations from robot sensors [RKH21, DLL22]. Previous works have successfully applied LLMs and VLMs in robot manipulation tasks within tabletop settings, guided by free-form natural language instructions [HLZ23, SZY24]. However, the applicability of this approach to more complex mobile manipulation setups, which entail exploration and navigation beyond fixed-base manipulation, remains largely unexplored.

In this work, we address Open-Vocabulary Mobile Manipulation (OVMM) [YRY23], which features multi-step mobile manipulation guided by free-form natural language instructions in open environments with only on-board sensing capabilities. Existing OVMM systems, such as OK-robot [LOP24], rely on open-loop integration of cutting-edge perception and robotic modules, which often struggle to adapt to unexpected scenarios or execution failures. To address this limitation, we propose COME-robot (**C**losed-loop **O**pen-vocabulary **M**obil**E** Manipulation), the first closed-loop framework that integrates GPT-4V [Ope23b], a state-of-the-art VLM, with a library of robust robotic primitive actions for real-robot OVMM. The closed-loop capability of COME-robot hinges on two pivotal designs: (i) **Actions as APIs**, where we implement the primitive actions of

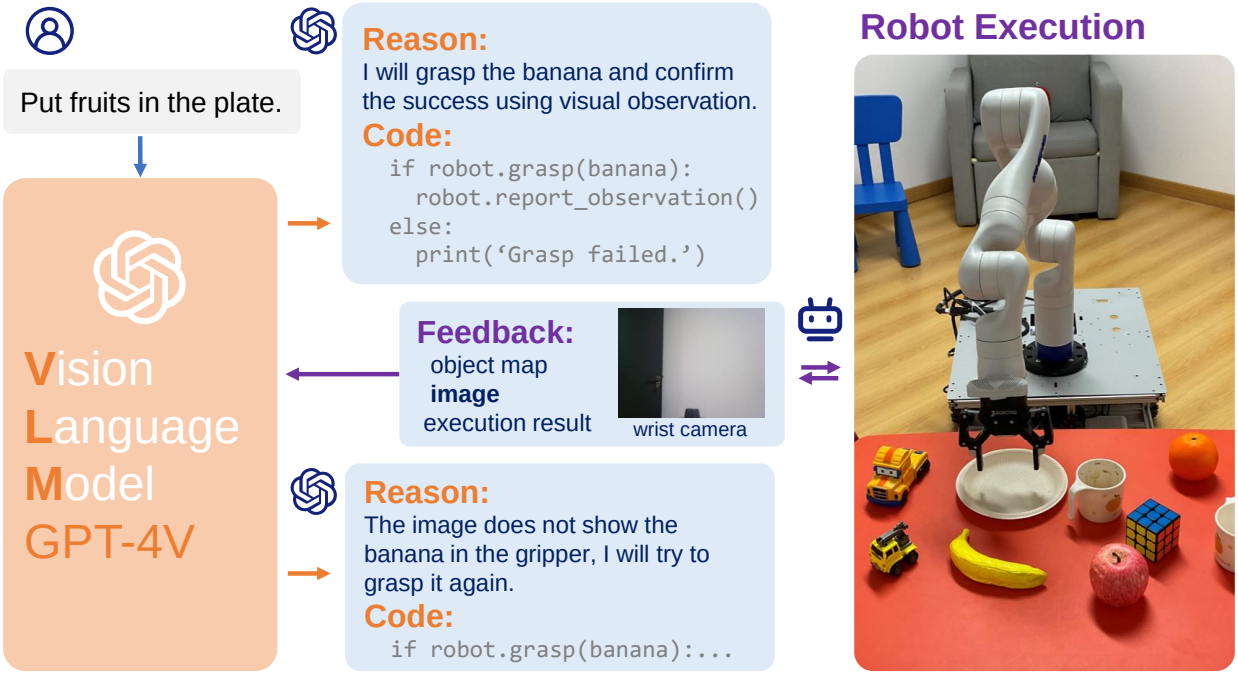


Figure 3.1: **A brief overview of COME-robot's workflow.** Given a task instruction, COME-robot employs GPT-4V for reasoning and generates a code-based plan. Through feedback obtained from the robot's execution and interaction with the environment, it iteratively updates the subsequent plan or recovers from failures, ultimately accomplishing the given task.

the robot as Python API functions, yielding multi-modal feedback such as observed images and reconstructed object maps upon execution. (ii) **GPT-4V as brain**, where we exploit the advanced multi-modal reasoning capabilities of GPT-4V to interpret language instructions, environment observations, and execution feedback, to generate Python code to command the robot by invoking action API functions. The overall workflow of COME-robot is shown in Fig. 3.1. Taking a language task specification from a human user, COME-robot operates by (i) querying GPT-4V for reasoning and code generation that reflects the robots' plan, (ii) executing the code on the real robot to obtain feedback, which is provided to GPT-4V in the next query. This iterative process continues until the goal is achieved or the program is manually terminated due to severe failures.

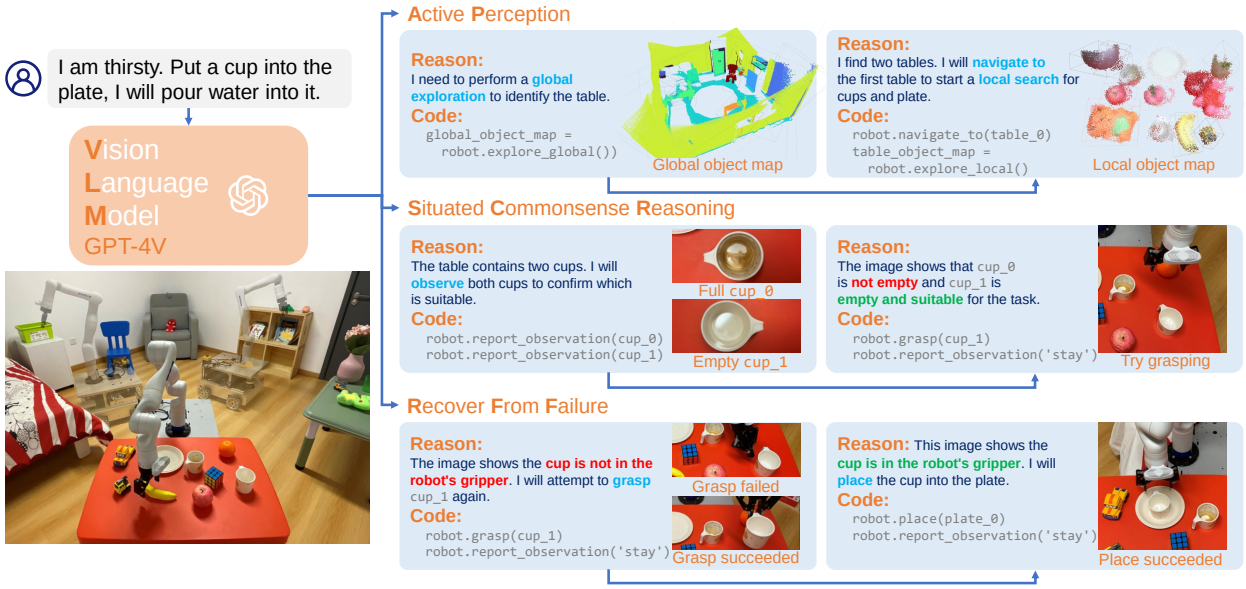


Figure 3.2: The unique properties of COME-robot: **Active Perception**, **Situated Commonsense Reasoning**, and **Recover from Failure**. Actions to be executed as reasoned by GPT-4V are highlighted in blue, identified failures are highlighted in red, and outcomes verified after observation or recovery are highlighted in green.

We conduct comprehensive real-robot experiments in a real-world bedroom to evaluate the effectiveness of COME-robot. We design a suite of 8 challenging OVMM tasks, with half of which entail manipulation around a table, while the other half further poses the challenge of navigating in the environment. We show with quantitative results that COME-robot significantly outperforms a powerful LLM-based baseline robot system [LHX23] in all tasks. We further conduct detailed case studies for experiments encountering various types of failures to provide a holistic view of COME-robot's ability to recover from failures. Our observations highlight that the closed-loop mechanism enables unique properties of COME-robot, as also shown in Fig. 3.2 to (i) actively perceive the environment with GPT-4V calling the perception APIs; (ii) perform situated commonsense reasoning, where GPT-4V interprets ambiguous instructions by grounding them in observed situations using its extensive commonsense knowledge; and (iii) recover from failures, where GPT-4V utilizes environment and execution feedback to detect, reason and rectify execution errors, ensuring

robust robot manipulation performance.

In summary, the contribution of this work is three-fold:

1. We present COME-robot, the first closed-loop OVMM framework that leverages GPT-4V for open-ended reasoning and replanning in real-world environments.

2. We devise an comprehensive library of action primitives for robot exploration, navigation, and manipulation, all implemented as Python API functions to provide multi-modal feedback for closed-loop OVMM.

3. We conduct comprehensive experiments on a real robot, showcasing state-of-the-art quantitative results alongside illustrative qualitative examples. In addition, we offer extensive discussions on emergent robot behaviors, such as failure recovery, exhibited by COME-robot.

3.2 Related Work

Mobile manipulation has long posed challenges in robotics research [Kha99], requiring the integration of both mobile base navigation and robot arm manipulation. Such an integration introduces new complexities. First, navigating in an extended workspace demands a sophisticated understanding of large-scale scenes for task execution, achieved through active, task-driven exploration and mapping from partial observations. Previous work has explored frontier-based [Yam97] and data-driven [ZMK17, CGK23] methods for scene exploration and object navigation. Efforts have also been made to build hierarchical scene representations [RGA20, GKM23, HZJ21, HZJ22] (*e.g.*, 3D scene graphs) for multi-resolution scene understanding. Second, many mobile manipulation tasks inherently involve planning long sequences of navigation and manipulation actions. This problem is addressed through hierarchical planning methods like task and motion planning [WMR10, JZW21a] or hierarchical reinforcement learning [GCS22, YCT23]. In this work, we address these challenges by developing a library of robust primitive actions for exploration, navigation, and pick-and-place, all following state-of-the-art practices.

Online replanning with closed-loop feedback is essential for long-term autonomous robot operation in the real world [KHD18]. Traditional robot replanning methods have focused on real-time collision-free motion planning in dynamic and uncertain environments [KL05, BPM11].In the context of long-horizon task and motion planning (TAMP), replanning has also been studied to recover from failure and handle partial observation [GPL20, CFK22]. However, these methods rely on hand-crafted modules and are less applicable to complex real-world settings. Recently, pre-trained foundation models [FTT23] have significantly enhanced robots' abilities to understand raw observations (*e.g.*, texts and images) and perform high-level reasoning and planning. Large Language Models (LLMs) have been particularly useful for reasoning and replanning based on textualized environment feedback [HXX23, YZY22, WCC23, WXJ23, DZA23, LBS23]. Vision Language Models (VLMs) [DLL22] further integrate visual understanding into the reasoning and planning loop [GWZ23, HLZ23, SZY24]. Specifically, VILA [HLZ23] and REPLAN [SZY24] leverage GPT-4V [Ope23b] to perform robot reasoning and closed-loop replanning directly on raw visual observations. In line with these works, we utilize GPT-4V for closed-loop reasoning and planning in mobile manipulation tasks that involve exploration, navigation, and manipulation.

Capabilities beyond reasoning and planning are enabled by LLMs and VLMs for robot agents. LLMs can generate computer programs to obtain observations and execute tasks by querying perception and action APIs [LHX23, SBM23, WXJ23]. Code-as-Policies [LHX23] presents a signature approach that commands a robot with LLM-generated Python code. On the other hand, VLMs endow robots with open-vocabulary visual understanding, allowing them to interpret and respond to free-form human language instructions. HomeRobot [YRY23] introduces the first benchmark for OVMM, where a mobile manipulator performs pick-and-place tasks with open-vocabulary language instructions. OK-Robot [LOP24] proposes a baseline system integrating state-of-the-art VLMs and robotic modules for the OVMM benchmark. In this work, we present COME-robot, which addresses OVMM tasks in a closed-loop manner by leveraging GPT-4V to iteratively generate code that invokes a library of primitive action APIs.

3.3 Method

In this section, we introduce COME-robot that tackles the challenging OVMM task. The OVMM takes an open-vocabulary language instruction from a human user as input, which specifies a mobile manipulation task, *e.g.*, "place all fruits on the plate". The robot is tasked to execute a sequence of exploration, navigation, and manipulation actions to achieve the task goal, given no prior knowledge of the environment. The COME-robot features implementing robot primitive actions as Python APIs, and using GPT-4V as the brain to perform reasoning and generate code to command the robot. We elaborate on our design of the action APIs in Section 3.3.1, describe our prompt engineering to utilize GPT-4V for reasoning and replanning in Section 3.3.2, and present the implementation details of API functions in Section 3.3.3.

3.3.1 Primitive Actions as APIs

We identify a library of primitive actions necessary for COME-robot to solve the OVMM task in the closed loop. We categorize them into two types of APIs and detail our design choice in the following. The full list of API functions can be found in Table 3.1.

Perception APIs Given the unknown and uncertain environment, COME-robot requires low-level primitives to explore the environment and draw visual observations for closed-loop operation. Specifically, robot navigation necessitates constructing a global object map of large furnitures (*e.g.*, tables and beds). On the other hand, a local object map with finer granularity (*e.g.*, small objects on a tabletop) is required for manipulation. We design perception APIs `explore_global` and `explore_local` for constructing the global and local object map, respectively. Collectively, the global and local object maps exhibit a two-level contact graph. To provide GPT-4V with visual feedback, we introduce an additional API `report_observation` that captures a 2D image using the robot's wrist camera. This API function is vital for checking the progress and outcome of action execution and enabling open-ended scene understanding using GPT-4V.

Function	Input	Return Type	Description
`explore_local`	`N/A`	ObjectMap	Explores the local area in front and identifies objects. Return an object map containing the objects.
`explore_global`	`N/A`	ObjectMap	The robot moves around and constructs a global, room-level map of the environment for navigation planning. Return an object map.
`report_observation`	`SceneObject | string`	2D Image	Takes a photograph and returns it. If given a `SceneObject` instance, the photo is captured from a close-up position targeting the specified object. If given a string, ``rest'' leads the robot to capture the image from its resting pose, and ``stay'' leads it to take the picture at its current state.
`navigate`	`SceneObject`	Boolean	Moves the robot to a specified object and facing towards it. Return success/failure.
`grasp`	`SceneObject`	Boolean	Executes an attempt to grasp the specified object. Return success/failure.
`place`	`SceneObject | location`	Boolean	Places a grasped object at a specified location, or near an object. Return success/failure.

Table 3.1: **Primitive action APIs for COME-robot's interaction with the environment.** All returned values serve as feedback.

Execution APIs OVMM tasks require COME-robot to navigate and manipulate with respect to target locations or specific objects in the environment. For navigation, we define an API `navigate` that drives the robot to approach an object that appears in the global object map. In terms of manipulation, we design a `grasp` API that commands the robot to grasp an object based on observed object point cloud, and a `place` API that allows the robot to place the object in hand on a receptacle or at a certain location.

3.3.2 Closed-loop Reasoning and Replanning with GPT-4V

Central to COME-robot's ability lies closed-loop reasoning and replanning with GPT-4V. Our intention is to have GT4-V to interpret feedback from primitive action APIs, and devise strategies and decisions with commonsense reasoning. To unleash the power of GPT-4V in reasoning, making high-level plans, and generating codes, we design the system prompt that incorporates the following aspects to elicit desired behaviors in real-robot planning:

- *Role*: A high-level description of the robot's role.
- *Feedback Handling*: Feedback handling guidelines for user queries and outputs of primitive action APIs.
- *Robot Setup*: Details about the robot's hardware capabilities, such as its arm, mobile base, and sensor suite.
- *APIs*: Detailed documentation of primitive action APIs.
- *Response Guidelines*: Instructions on Chain-of-Thought [WWS22] reasoning and JSON format specifications.
- *Tips*: Useful tips for producing robust robot behaviors.

We provide a system prompt snapshot in Fig. 3.3. The system prompt and user query are fed into GPT-4V, which generates Python code executed by the robot. Feedback from API execution is then relayed back to GPT-4V, informing its future reasoning and planning processes.

3.3.3 Implementation Details

Real-robot Setup We use a mobile manipulator that comprises four major components: (i) A four-wheeled differential drive mobile base equipped with an RGB-D camera, a Lidar and an IMU unit; (ii) A 7-DoF Kinova Gen3 robot arm with a Robotiq parallel gripper and a RGB-D camera on its wrist; (iii) An onboard PC that directly interfaces the sensors and controls the robot, and communicates with a desktop server to send sensor observations and receive commands; and (iii) A remote desktop server that runs GPT-4V and other heavy perception modules. We use the ROS [QCG09] framework for communication between the robot and computers.

Global Exploration and Navigation We use the RTAB-Map [LM19] package to fuse IMU odometry, Lidar data, and RGB-D streams for localization. For the `explore_global` API, we follow [CGK23] to incrementally build a 2D obstacle map from RGB-D streams and adopt a frontier-based strategy to propose the next target location to visit. Then we employ a Fast Marching Method (FMM)-based path planner [Set96] to navigate the mobile base to the target location.

Task : You operate as a brain for a robot that is specialized in planning within a novel scene. You will first receive a request from a user, then you use available **APIs** of the robot to perceive the environment, plan actions, execute actions and get **feedback**. You will change your plans if necessary.
Feedback handling:
Each step, the Python code generated by you will be executed on the robot, with the outputs returned back to you, including those from `print` and raised exceptions.
Robot setup:
The robot has a movable base and… *[details about robot hardware setup]*
Documentation:
class `Robot`, a robot that can move around and interact with the environment. Methods:
- `explore_global(self, )->ObjectMap:…` *[details]*
- *[more methods]*

class `ObjectMap`, an object map that contains some objects. Methods:
- `__getitem__(self, key:str)->SceneObject:…` *[details and an example]*
- *[SceneObject documentation]*

Response guidelines: You should always think **step by step**, first give some high-level thoughts and reason what to do next, then generate the concrete Python code. You should always response in **JSON** format, with a "**reason**" field and a "**code**" field. *[more details about JSON format and an example]*
Tips:
- You can use the robot's vision system to verify the grasping and placing results.
- Only navigate when necessary. If already positioned suitably, navigation is not needed.
- Perform tasks in a step-by-step manner, verifying each step before continuing to the next.
- *[some more tips…]*

Figure 3.3: **A snapshot of COME-robot's system prompt.**

As the robot explores the environment, we use Grounded SAM [RLZ24] to detect and segment objects on RGB images, and obtain per-frame object point clouds by cropping depth images with object masks. The per-frame results are then fused into a global object map following ConceptGraphs [GKM23], which maintains the category, instance id and point cloud of objects. Note that we only keep large furniture objects in the global object map. For the `navigate` API for object navigation, we first calculate a target pose that faces the target object and within a short distance, and then navigate the robot to the target pose leveraging the same path planner as in exploration.

Local Exploration and Manipulation For the `explore_local` API that explores a local region for manipulation, RGB-D images are captured from the robot arm's wrist camera to construct

a local object map. Specifically, we select three camera poses based on manual-defined heuristics to ensure comprehensive coverage of objects in observations within the robot arm's workspace. Then we construct the local object map similar to `explore_global`.

For API `grasp`, we retrieve the target object's point cloud from the local object map and generate 6-DoF grasp poses using Contact-GraspNet [SMT21]. After filtering to retain up to 20 high-confidence candidates, ideally favoring top-down grasps, we use the RRT [LaV98]-based motion planner from MoveIt [CSC14] to calculate a collision-free trajectory for the robot arm to reach these poses, treating other object point clouds as obstacles. The grasping process repeats for all candidate poses until a complete grasp action (*i.e.*, move to the grasp pose and close the gripper) is planned and executed, starting with the highest-confidence candidate. If no reachable grasp pose is found, the API call returns "False".

For the `place` API, we similarly plan collision-free trajectories to position the gripper 15cm above the target, then execute the trajectory and release the gripper to place the object. The API function returns "False" if no feasible trajectory exists.

3.4 Experiments

This section provides details on experiment task settings and the implementation details of baseline methods. We demonstrate the effectiveness of our system through thorough comparisons with baseline methods and highlight the significance of our design by systematically analyzing execution trials of COME-robot on these tasks.

3.4.1 Experimental Setup

Baseline We consider a recent method, Code as Policies (CaP) [LHX23] as our baseline. CaP leverages LLM to generate code to command robots in complex tasks. As the original CaP method assumes a fully observable tabletop environment, we augment CaP with our designed exploration

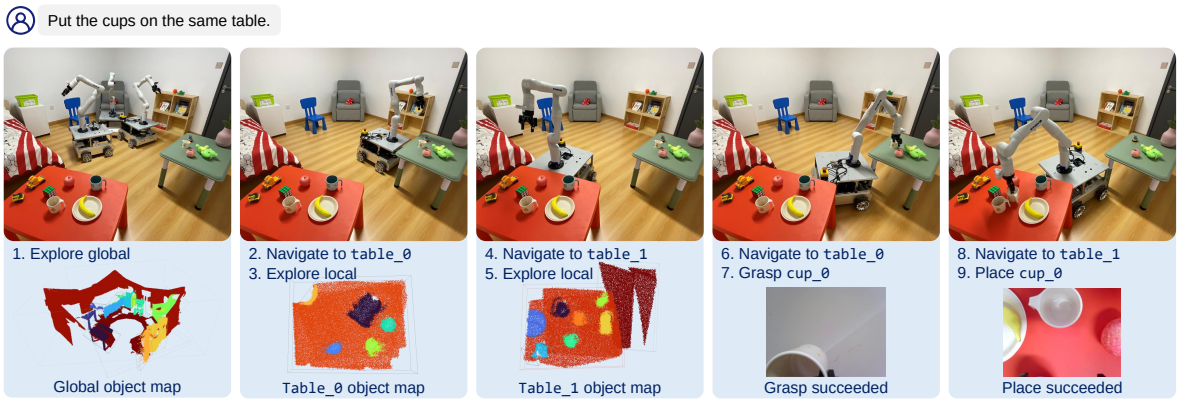


Figure 3.4: **A step-by-step visualization of COME-robot's task execution in GATHER CUPS.** With the query "Put the cups on the same table." The robot builds a global object map and locates two tables. It then navigates to `table_0`, explores locally and identifies one cup on the table. It continues to inspect `table_1` and identifies two cups. With situated commonsense reasoning, COME-robot decides to move the cup from `table_0` to `table_1` as it is more efficient. It thus navigates back to `table_0`, grasps the cup, and verifies the success of grasp with the wrist camera. Finally, it navigates back to `table_1` to place the cup down. With the placement once again verified, the task is considered complete.

and navigation API functions to build a strong baseline for comparison, referred as CaP*.

Task Design We meticulously design 8 challenging tasks with various horizons in a real-world room to evaluate COME-robot's capability for OVMM. The room contains diverse furniture, including several tables, a bed, a sofa, and various objects. We mainly consider two experimental settings, the tabletop (4 tasks) and the mobile manipulation (4 tasks). We design tasks that aim to verify different capabilities of the COME-robot framework. Specifically, the tabletop tasks are:

A1 PLACE FRUIT: The table is initialized with several fruits, a plate, and other items. The task is to put fruits onto the plate with clear instructions to test the robots' *basic execution and instruction-following* ability.

A2 FRUIT AMONG CUPS: The table is initialized with a fruit, several cups, and other items. The

Table 3.2: **Quantitative results on tabletop manipulation.**

Tabletop Task	CaP*		COME-robot		
	SR	SSR	SR	SSR	RR
PLACE FRUIT	5 / 10	30 / 40	**7 / 10**	**34 / 40**	4 / 7
FRUIT AMONG CUPS	6 / 10	13 / 20	**8 / 10**	**18 / 20**	1 / 3
PREPARE CUP	4 / 10	12 / 20	**8 / 10**	**17 / 20**	7 / 10
TIDY TABLE	4 / 10	43 / 58	**7 / 10**	**54 / 60**	5 / 9
Total	19 / 40	98 / 138	**30/40**	**123 / 140**	17 / 29

Table 3.3: **Quantitative results on mobile manipulation.**

Mobile Task	CaP*		COME-robot		
	SR	SSR	SR	SSR	RR
MOVE TOY	2 / 5	13 / 20	**3 / 5**	**17 / 20**	2 / 4
TRANSFER ALL TOYS	1 / 5	24 / 42	**2 / 5**	**30 / 42**	1 / 4
MOVE CUP AND TOY	1 / 5	17 / 30	**4 / 5**	**27 / 30**	4 / 5
GATHER CUPS	2 / 5	22 / 33	**4 / 5**	**27 / 30**	7 / 10
Total	6 / 20	76 / 125	**13/20**	**101 / 122**	14 / 23

task is to place a fruit in the middle of all cups with instructions like "Put the fruit in the middle of the cups". This tests robots' *understanding of spatial concepts* in instructions.

A3 PREPARE CUP: The table is initialized with an empty cup, a used cup, a plate, and other items. The task is to pick up the clean, unused cup and put it onto the plate. We provide indirect instructions like "Put a cup onto the plate, I will pour water into it.", requiring *commonsense reasoning* for identifying the targeting object.

A4 TIDY TABLE: The table is cluttered with storage boxes and objects (*e.g.*, fruits, toys, *etc.*). The task is to place these objects into their corresponding storage boxes. We provide no direct instructions to robots, using instructions like "Can you help me tidy up the table?", to further test robots' *concept understanding and reasoning capability with free-form instruc-*

tions.

For mobile manipulation, we consider the following tasks:

B1 MOVE TOY: The task is to move the toy from the table to the bed with clear instructions. This task tests the *basic mobile manipulation capabilities* of robots.

B2 TRANSFER ALL TOYS: The task is to move scattered toys from different tables to the sofa, with instructions like "Place all toys on tables to the sofa.". This targets robots' *object search capability* in the environment.

B3 MOVE CUP AND TOY: The task is to find a cup with specific visual attributes, place it on a plate, and move a toy to the bed, following instructions like "Put the blue cup on the plate and doll to the bed". This evaluates robots' ability in *sequential tasks and visual reasoning*.

B4 GATHER CUPS: The task is to get all water cups from multiple tables and place them on a single table without specifying which table, *i.e.*, using instructions like "Put cups onto the same table.". This assesses robots' ability to *make efficient plans* when selecting the target table.

Experiment Setting For tabletop tasks, we conduct 10 experiment trials for each task. We adjust the scene configuration between trials by introducing variations of objects' types, their arrangements, and quantities on the table. For mobile manipulation tasks, we conduct 5 trials for each task. Similarly, we rearrange the location of objects and furniture in the room as variations between trials.

Metrics We report the success rate (SR) of goal completion and the step-wise success rate (SSR) of action execution for all models. Each execution API call is considered one step (*i.e.*, excluding perception APIs like `report_observation`), and we count the number of successful ones over all calls for SSR. Different planned paths and methods may require varying numbers of steps for the same task, especially considering COME-robot's replanning mechanism. Additionally, to unveil COME-robot's ability to recover from failure, we report the recovery rate (RR) of COME-robot by

tallying all replanned executions and the successful ones within these executions.

3.4.2 Experimental Results and Analyses

As shown in Table 3.2 and Table 3.3, COME-robot achieves consistent and significant improvements in goal completion for both the tabletop and the mobile manipulation setting. Specifically, COME-robot achieves an overall success rate of 75% (30/40) on the tabletop setting, outperforming the CaP* baseline (47.5%, 19/40) by 27.5%. Similarly, under the mobile manipulation setting, COME-robot achieves a success rate of 65%, significantly outperforming baseline (30%, 6/20) by 35%.

Additionally, the ability of COME-robot to recover from failures significantly enhances its stepwise success rate by identifying the failure step and replanning, demonstrating a higher performance of $123/140$ compared to CaP*'s $98/138$ and $101/122$ compared to $76/125$. This improvement also contributes to the higher overall success rate. These quantitative results validate that COME-robot, equipped with the ability to replan using closed-loop feedback, effectively identifies and corrects errors encountered during task execution. This capability facilitates task execution and goal achievements in challenging real-world tasks.

3.4.3 Failure Recovery

As shown in Table 3.2 and Table 3.3, we demonstrate the effectiveness of failure recovery from both the number of recovering steps and also the high recovery rate. In this section, we provide analysis to systematically categorize COME-robot's failure cases and highlight how COME-robot recovers from such failures utilizing the VLM feedback.

Perceptual Failures Perception failures are primarily caused by errors in detection during exploration. COME-robot can use `report_observation` function for close inspection and utilize visual feedback for solving the missed or wrong detection problem. For missed detections, COME-

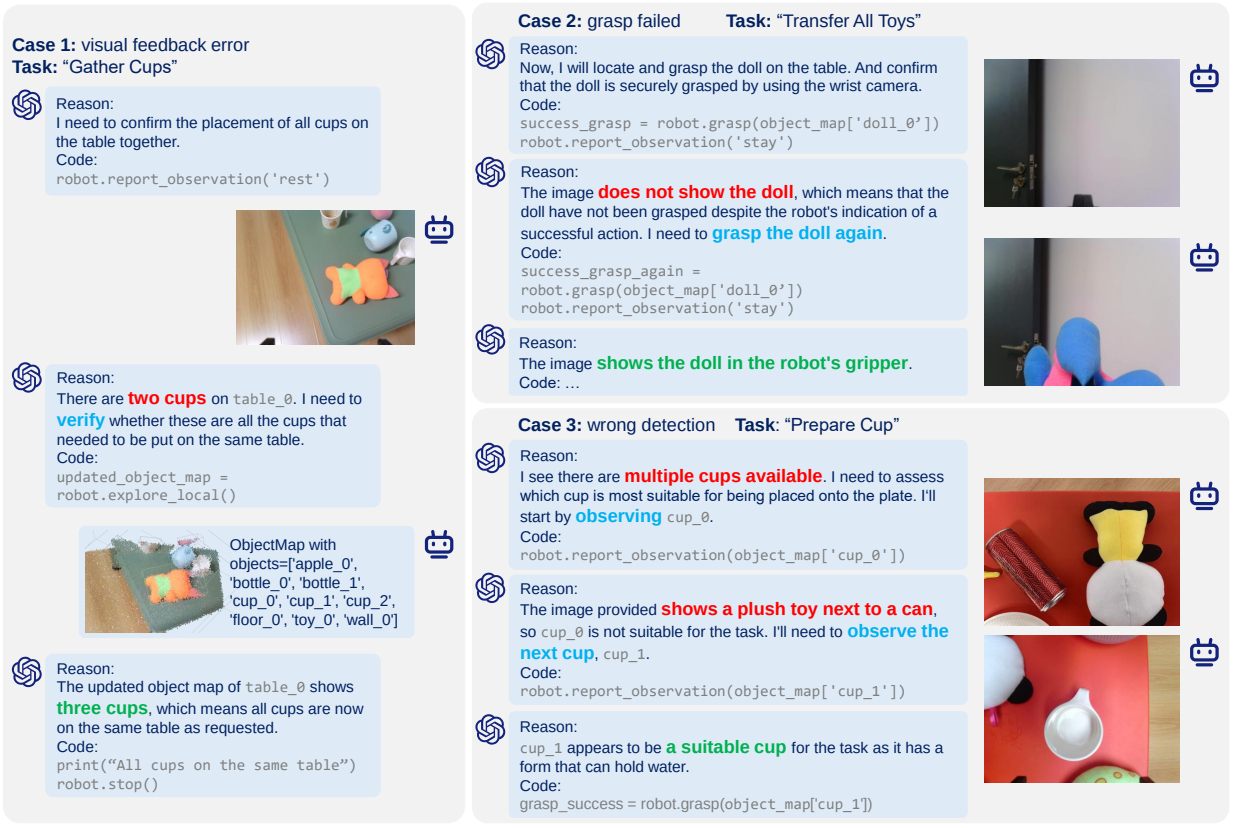


Figure 3.5: **Three examples of recovery from failures.** Case 1 illustrates recovery from a visual feedback error with the VLM by updating the object map. Case 2 demonstrates recovering from a failed grasp attempt by trying again. Case 3 describes a scenario of false positive detection, and recover through visual feedback.

robot directs perception modules to rebuild the local object scene graph and re-detect the missing object, achieving a 100% recover rate as shown in Table 3.4. For wrong detections, COME-robot utilizes GPT-4V to conduct a verification step for detected objects. For example, in case 3 of Fig. 3.5, when the `explore_local` function detects multiple candidate cups, COME-robot verifies each cup with image observations and finds that `cup_0` is actually a doll that is wrongly detected as cup and not related to the task. Though this verification process can help mitigate the problem, it is still error-prone to incorrect predictions, leaving 6 falsely detected objects after verification, with three of which lead to task failure as shown in Table 3.4. Other errors stem from

mistakes in visual analysis, which are due to blurred images or issues inherent to the VLM. For example, in case 1 of Fig. 3.5, when GPT-4V attempts to confirm the success of the placement, it only sees two cups because the image does not completely capture the cups, leading to a misjudgment. However, COME-robot corrects this error by conducting another local exploration and discovering that there are actually three cups on the table.

Execution Failures COME-robot's GPT-4V-based planning method may sometimes generate incorrect plans or invalid API calls, such as attempting to place an object without prior grasping or calling the navigation function with an object name instead of an object. For these errors, COME-robot verifies the generated plan and code, and triggers exceptions during execution, providing explicit feedback indicating the missing step or wrong function call for GPT-4V to rectify the plan. For actual execution, the primary source of failed execution is caused by unsuccessful grasps. Grasping failures are primarily due to the impractical position the robot navigates to that significantly constrains its space for manipulation (*e.g.* corner of the table, or close to the wall). COME-robot can use the same feedback mechanism to identify and rectify execution missteps, such as failed grasps or placements, by executing corrective actions based on visual confirmations, thereby enhancing task success rates despite initial setbacks. The case 2 of Fig. 3.5 shows an example of recovering from grasp failure.

3.4.4 Discussions

Is commonsense reasoning using LLMs enough for mobile manipulation with open-ended instructions? LLMs possess robust commonsense reasoning capabilities that allow robots to perform various generalized tasks based on high-level and abstract human instructions (*e.g.*, in the TIDY TABLE (A4) task). However, we argue that completing these tasks requires both the commonsense knowledge provided by LLMs and the ability to interactively explore and update scene information. Taking the PREPARE CUP (A3) task as an example, when multiple cups are detected, the robot must identify the most suitable one for the task. Crafting a plan solely based on

Table 3.4: **Statistics of COME-robot's failure and recovery.** We use "DF." to denote failures that lead to direct failure of the task, "R-RR." to denote the recovery rate of the re-planning steps.

Failure Type	Reason	DF.	R-RR.	Total
Perception	false positive	3	5/5	5/8
	missed detection	1	1/1	1/2
	visual feedback error	2	1/1	1/3
Execution	API call error	3	5/5	5/8
	grasp failed	1	17/23	17/24
	place failed	3	3/3	3/6
	navigation failed	1	0/0	0/1
Total		14	31/38	31/52

the instruction becomes challenging without understanding the status of all cups.

Why is replanning with closed-loop feedback important in robot manipulation tasks? Compared to tabletop tasks, mobile manipulation involves longer sequences, requiring the robot to first explore the room and then shuttle between furniture to complete cross-furniture manipulation tasks. As shown in Table 3.3, mobile manipulation tasks require a significantly higher average execution step. Meanwhile, the long execution sequence unveils the effectiveness of replanning in COME-robot compared to CaP* as long sequences increase the likelihood of execution failure (*e.g.*, in MOVE CUP AND TOY (B2) and TIDY TABLE (A4) tasks). Replanning with closed-loop feedback enables COME-robot to detect failures and attempt to recover from them, thereby increasing the success rate of actions and reducing the likelihood of task failure due to intermediate setbacks.

3.5 Conclusion

In this chapter, we present COME-robot, a novel closed-loop framework that integrates the vision-language model GPT-4V with a library of robust robotic primitives to enable open-vocabulary mobile manipulation in real-world environments. Through extensive real-robot experimental analyses, we showcase COME-robot's superior ability to interpret open-ended instructions, actively acquire and reason over multi-modal feedback, and adaptively recover from perception and execution failures. Leveraging GPT-4V's powerful reasoning capabilities, COME-robot achieves unprecedented flexibility and intelligence in handling challenging OVMM tasks. Our framework represents a significant step towards developing autonomous robots capable of operating effectively in complex, unstructured real-world settings. We believe the insights gained from this work will inspire future research on integrating foundation models with robotic systems to advance robot intelligence and autonomy.

CHAPTER 4

Task and Motion Planning with Large Language Model and Motion Failure Reasoning

Conventional TAMP approaches rely on manually designed planning domains and interfaces that connect symbolic task planning with continuous motion generation. These domain-specific and labor-intensive modules are limited in addressing emerging tasks in real-world settings. This chapter presents LLM^3, a novel LLM-based TAMP framework featuring a domain-independent interface. Specifically, we leverage the internal world knowledge and powerful reasoning and planning capabilities of pre-trained LLMs to propose symbolic action sequences and select continuous action parameters for motion planning. Crucially, LLM^3 incorporates motion planning feedback through prompting, allowing the LLM to iteratively refine its proposals by reasoning about motion failure. The materials in this chapter have been published in [WHJ24].

4.1 Introduction

Sequential manipulation planning is an essential capability for robots to autonomously perform diverse tasks in complex environments. Executable motions must be effectively generated for robots to achieve long-term task objectives, requiring efficient planning algorithms for responsive operation and reasoning capabilities to anticipate environmental changes. Task and Motion Planning (TAMP) formulates a methodology that hierarchically decomposes planning into two stages: the high-level symbolic task planning stage reasons over long-horizon abstract action sequences, and the low-level continuous motion planning stage computes feasible trajectories subject to ge-

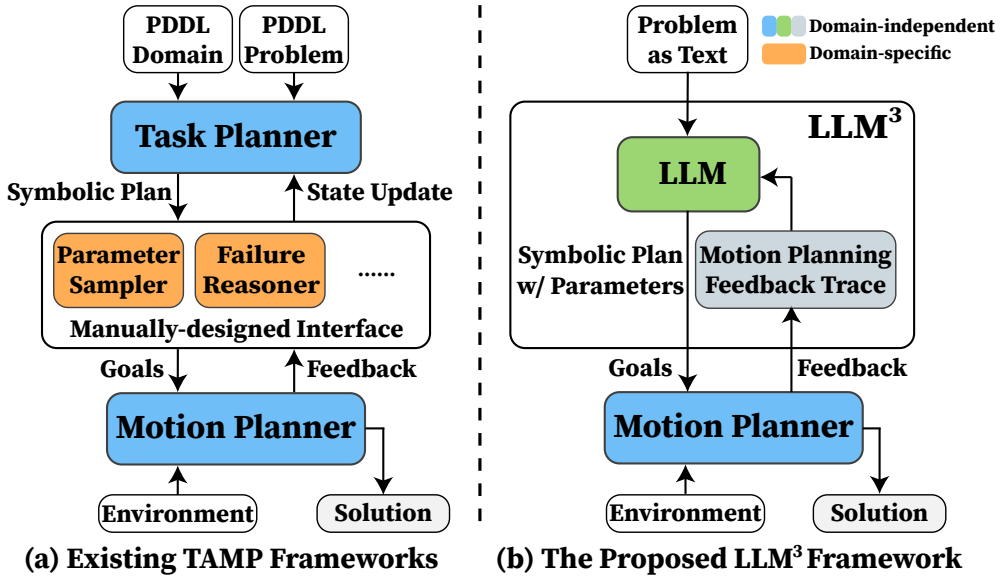


Figure 4.1: **The proposed LLM^3 framework.** (a) Traditional TAMP frameworks rely on manually designed, domain-specific modules for interfacing between task and motion planners. (b) In contrast, we leverage a pre-trained LLM to iteratively propose refined plans and action parameters, by reasoning on motion planning failures.

ometric constraints. In recent years, TAMP has enabled significant advances [DKC16, TAS18, GCH21, JZW21b, JNZ22, SLJ23]. However, a persistent challenge remains to properly interface between the task planner and the motion planner to efficiently solve TAMP, *i.e.*, generating action sequences that satisfy both symbolic task goals and continuous motion constraints.

Traditional TAMP approaches often rely on manually designed modules to interface between symbolic and continuous domains, as depicted in Fig. 4.1(a). These modules serve two key roles. First, they act as action parameter samplers that generate real-valued parameters for symbolic actions. These parameters provide numerical goals to the motion planner. Selecting appropriate action parameters, *e.g.*, in the object rearrangement task, a suitable 2D target location (x, y) for action `Place(object)`, is crucial for the motion feasibility of actions and the efficiency of TAMP. While previous works propose to learn heuristic parameter samplers from data [CHG16, WGK18], they are tailored to specific domains and lack generalizability. Second, these modules

implement mechanisms to incorporate motion failure into the task planner to generate improved action plans, *e.g.*, by updating the symbolic state [SFR14]. However, they usually require domain-specific design by human experts. In summary, these modules are domain-specific and require substantial manual effort to design, which hinders generalizability to novel environments.

Recent Large Language Models (LLMs) [ZZL23] pre-trained on web-scale text data have demonstrated emergent capabilities in reasoning [KGR22] and planning [HAP22] when provided with in-context prompts [BMR20, DLD22]. Pre-trained LLMs can perform task planning [HAP22, ABB22, HXX23], generate continuous parameters [MXF23], and reason on environment feedback [YZY22, HXX23]. Our intuition is that pre-trained LLMs could provide a general and domain-independent approach to interfacing between symbolic and continuous domains for TAMP, eliminating the need to design domain-specific modules manually.

In this work, we present **LLM^3** (**L**arge **L**anguage **M**odel-based Task and **M**otion Planning with **M**otion Failure Reasoning), an LLM-powered TAMP framework that reasons over motion planning feedback for effective planningFig. 4.1(b). Specifically, LLM^3 employs a pre-trained LLM to (i) propose symbolic action sequences towards the task goal, (ii) generate continuous action parameters that lead to feasible motion, and (iii) reason over motion planning feedback to iteratively refine the proposed symbolic actions and parameters. This framework offers several key advantages over traditional TAMP approaches. First, it does not require manually designed symbolic domain files for task planning, instead leveraging the knowledge encoded in the LLM to propose symbolic actions. Second, it uses the LLM as a domain-independent informed parameter sampler to generate continuous action parameters, which benefits from the implicit heuristics of the LLM [MXF23]. Third, its reasoning over motion planning feedback is independent of the specific choice of planner. Crucially, we categorize and organize the possible motion planning feedback to feature two major motion failure modes, *i.e.*, collision and unreachability. Such motion planning feedback allows LLM^3 to refine the generated action sequences and parameters in a more targeted way, and find a feasible TAMP solution with fewer planning iterations and motion planning queries.

We evaluate LLM^3 in a simulated tabletop box-packing task, which poses challenges in rea-

soning about potential failure modes, collisions, and unreachable areas, throughout the sequential manipulation planning problem. Quantitative results demonstrate the effectiveness of LLM^3, with ablation studies verifying: (i) reasoning over motion feedback significantly improves success rates and planning efficiency, and (ii) the LLM-based parameter sampler is substantially more sample efficient than a random sampler. Furthermore, we conduct real-robot experiments to show that LLM^3 can be applied to real-world problems.

In summary, our contributions are threefold:

1. We introduce LLM^3, the ***first*** TAMP framework that holistically employs a pre-trained LLM as a domain-independent task planner, informed action parameter sampler, and motion failure reasoner.

2. We categorize and organize the feedback from the motion planner, which enables LLM^3 to efficiently identify and resolve planner-independent motion failures through targeted refinement of actions and parameters.

3. We conduct comprehensive experiments in both the simulation and the real world to demonstrate the effectiveness of LLM^3 in solving TAMP problems.

4.2 Related Work

Task and Motion Planning Conventional TAMP approaches employ a high-level task planner to generate symbolic action sequences and a low-level motion planner to generate motion trajectories. The task planner requires pre-designed symbolic planning domains represented in formatted representations, such as PDDL. Significant efforts have been made to develop manually engineered modules that interface the task planner and motion planner, such as incorporating motion-level constraints into task planning [GLK20, JZW21b], making approximations at the motion level [HN11, Tou15], and designing specialized communication modules [SFR14]. However, manually defining task planning domains and interface modules to fully capture real-

world complexity is impractical. Furthermore, as the action space grows, searching for geometrically feasible symbolic action sequences becomes computationally challenging without effective heuristics [GLK20]. Recent work has explored data-driven heuristics to improve TAMP efficiency [CHG16, WDS19, NMB21, YGL23], but these domain-specific heuristics lack generalizability across domains. In this work, we employ a pre-trained LLM as both the task planner and the interface between task and motion. We expect that semantic knowledge in the LLM can provide domain-independent heuristics for TAMP.

Robot Planning with LLMs Recent Large Language Models (LLMs) [ZZL23] encode vast world knowledge and exhibit the emergent capability for planning [HAP22, LPP22] through few-shot or zero-shot in-context learning [BMR20, DLD22, DLD22]. Pre-trained LLMs have been applied for task planning of robots or embodied agents [ABB22, HXX23, LHX23, SBM23, WCC23, WXJ23, YZY22, HZZ24]. Notably, Inner Monologue [HXX23] takes in textualized environment feedback and generate actions to execute, while ReAct [YZY22] further advanced this closed-loop approach by integrating reasoning and acting. Voyager [WXJ23] and DEPS [WCC23] focus on developing open-ended embodied agents that iteratively replan based on execution failure in PC games. Furthermore, LLMs have been applied to solve TAMP problems [DZP23, CAZ23]. Specifically, LLM-GROP [DZP23] employs a pre-trained LLM to instantiate symbolic goals and continuous object placements for semantic object rearrangement, which are then fed into a classical TAMP planner. AutoTAMP [CAZ23] leverages an LLM to translate natural language task specifications into a formal language processable by off-the-shelf TAMP algorithms. Our usage of LLMs in TAMP is inspired by many of the above works; however, the major difference is that we leverage the LLM as the core component of our TAMP framework.

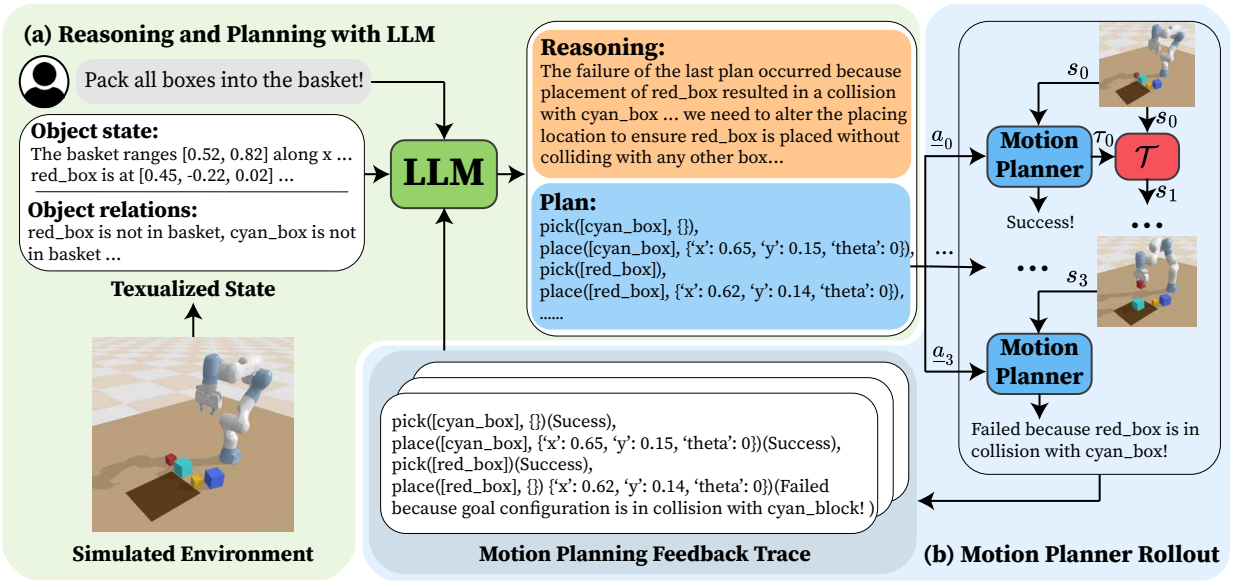


Figure 4.2: **System diagram of the proposed LLM^3 framework.** (a) We show an example of utilizing a pre-trained LLM for reasoning and generating action sequences. (b) The feasibility of the proposed action sequence is verified by rollout with a motion planner and transition function $\mathcal{T}$. The motion planning feedback is saved into a trace and provided to the LLM in the next iteration.

4.3 Preliminaries and Problem Setting

4.3.1 Task and Motion Planning

A *TAMP* problem is a tuple $< \mathcal{O}, \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, g >$, where:

- $\mathcal{O}$ is the set of objects in the environment.
- $\mathcal{S}$ is the state space factorized with respect to objects $\mathcal{O}$, where a state $s \in \mathcal{S}$ comprises the state of all objects, *e.g.*, their 3D positions and dimensions.
- $\mathcal{A}$ is the set of primitive actions with low-level execution handled by a motion planner. A primitive action $a \in A$ is parameterized by object variables $\bar{o}$ and continuous parameters θ, which can be instantiated with specific objects $\underline{o} = (o_1, ..., o_m)$ and parameter values to

produce a ground action $\underline{a}$, *e.g.*, $\texttt{Place}(\texttt{block}, [0.1, 0.2])$. The parameters θ provide the goal for the motion planner; we say $\underline{a}$ is feasible when the motion planner has a solution, *i.e.*, a collision-free trajectory τ. We denote a feasible action by $\underline{a}(\tau)$.

- $\mathcal{T}$ is the state transition function that outputs the next state s_{t+1} after executing an action $\underline{a}_t(\tau_t)$ at a state s_t, *i.e.*, $s_{t+1} = \mathcal{T}(s_t, \underline{a}_t(\tau_t))$. We assume that $\mathcal{T}(s_t, \underline{a}_t(\tau_t)))$ can be evaluated with a black-box simulator.

- s_0 is the initial state that follows $s_0 \in \mathcal{S}$.

- g is the goal function $g: \mathcal{S} \rightarrow \{0, 1\}$ that checks whether the task goal is achieved at state s.

The objective of TAMP is to derive a sequence of feasible actions $(\underline{a}_0(\tau_0), \underline{a}_1(\tau_2), ..., \underline{a}_T(\tau_T))$, such that $g(s_{T+1}) = 1$ and $s_{t+1} = \mathcal{T}(s_t, \underline{a}_t(\tau_t)))$, where $t = 0, 1, ..., T$.

A common strategy to solve the TAMP problem is known as "search-then-sample" [SFR14, DKC16, GCH21], which alternates between generating symbolic action sequences through backtracking search and sampling the continuous action parameters until a feasible plan is found to reach the goal. However, a naive search-then-sample task and motion planner is usually inefficient, as verifying the action feasibility requires invoking the computationally expensive motion planning process. To mitigate this complexity, researchers have crafted symbolic domains [SFR14, SCK23] to prune infeasible symbolic action sequences, or learned heuristic samplers to sample action parameters [WGK18, FGE23] that lead to feasible plans. Here, we leverage pre-trained LLMs as both a task planner and a heuristic sampler to generate symbolic action sequences and action parameters.

4.3.2 Planning as Sequence Prediction

Recently, planning problems have been formulated as sequence prediction to avoid the computationally expensive search process [DHT20, JLU23]. In particular, pre-trained LLMs are employed to generate discrete actions [HAP22, LPP22] and continuous parameters [LHX23] in an auto-

regressive manner when provided in-context prompts [DLD22]. Formally, with textualized initial state s_0, goal g and additional context c, we have:

$$\begin{aligned} \underline{a}_{0:T} &= \arg\max \; p_{LM}(\underline{a}_{0:T}|s_0, g, c) \\ &= \arg\max \; p_{LM}(\underline{a}_0|, s_0, g, c) \prod_{t=1}^{T} p_{LM}(\underline{a}_t|\underline{a}_{0:t-1}, s_0, g, c), \end{aligned} \tag{4.1}$$

where p_{LM} is the generative probability of a pre-trained language model. Following this scheme, we use pre-trained LLMs to propose sequences of symbolic actions and continuous action parameters based on the initial state, goal, and trace of motion planning feedback. As the proposed action sequences may be infeasible at the motion level, we validate the feasibility of proposed action sequences with a motion planner, the state transition function $\mathcal{T}$ and the goal g. We iterate these two steps until a feasible plan is found.

4.4 Method

We introduce LLM^3, a TAMP framework that leverages a pre-trained LLM to reason on motion failure and generate iteratively refined symbolic actions and continuous action parameters. The system diagram of LLM^3 is shown in Fig. 4.2. Below, we elaborate on the overall framework, reasoning and planning with the pre-trained LLM, and the designed motion planning feedback.

4.4.1 The LLM^3 Framework

As shown in Algorithm Algorithm 1 and Figure 4.2, the LLM^3 framework iterates between: (i) reasoning on previous motion failure and generating an action sequence (*i.e.*, symbolic actions and continuous parameters) with a pre-trained LLM, and (ii) verifying the feasibility of the action sequence with a motion planner. At each planning iteration, the LLM takes the current state s and the trace of motion planning feedback $trace$, and outputs the reasoning for the previous motion failure $reason$ and an action sequence llm_plan to solve the TAMP problem (see Section 4.4.2). The motion planner then attempts to find a collision-free motion trajectory for each action $\underline{a} \in llm\_plan$

Algorithm 1: LLM^3 for TAMP

Input : pre-trained LLM with prompt template LLM, state transition function $\mathcal{T}$ with motion planner MP, goal function g, initial state s_0

Output: success indicator $success$, a sequence of feasible actions $plan$

$plan \leftarrow []$, $success \leftarrow$ False, $feedback \leftarrow []$, $trace \leftarrow []$, $s \leftarrow s_0$, $iter \leftarrow 0$

// main planning loop

while *not* $success$ *and* $iter < N_{max}$ **do**

  // reason and plan with LLM

  $reason, llm\_plan \leftarrow LLM(s_0, trace)$, $iter \leftarrow iter + 1$

  // rollout llm_plan from s with motion planner

  foreach $\underline{a} \in llm\_plan$ **do**

    // call motion planner to verify feasibility of $\underline{a}$

    $\tau, mp\_feedback \leftarrow MP(s; \underline{a})$

    $feedback.append((\underline{a}, mp\_feedback))$

    // terminate rollout if action $\underline{a}$ is infeasible

    if τ *is* $None$ **then**

      break

    // update state s and add $\underline{a}$ to $plan$

    $s \leftarrow \mathcal{T}(s, \underline{a}(\tau))$, $plan.append(\underline{a}(\tau))$

  // check whether s satisfies goal

  $success, task\_feedback \leftarrow g(s)$

  // when the plan is feasible but doesn't reach goal

  if *not* $success$ **then**

    // record $task\_feedback$, add aggregated feedback to $trace$, and clear aggregated feedback

    $feedback.append(task\_feedback)$

    $trace.append(feedback)$, $trace \leftarrow trace[-k:]$, $feedback \leftarrow []$

return $success$, $plan$

sequentially. We synthesize motion planning feedback (see Section 4.4.3) for each motion planner query and aggregated the feedback for all actions in llm_plan. The planning iteration ends with failure when an action has no feasible motion or the action sequence fails to reach the goal. The aggregated feedback is then added to a trace $trace$, which is maintained throughout the life cycle of the framework with a maximum size k. In the next planning iteration, the trace is fed into the LLM to generate motion failure reasoning and another action sequence that improves on the previous one. This process repeats until a generated action sequence reaches the goal with no motion failure or the maximum number of attempts is exceeded. Overall, the LLM^3 framework can be regarded as a search-then-sample TAMP planner that generates action sequences with incrementally improved quality, guided by the intrinsic heuristics of the pre-trained LLM and the previous motion failure. By design, we expect LLM^3 to exhibit superior efficiency compared to unguided planners that sample action parameters randomly.

4.4.2 Reasoning and Planning with pre-trained LLM

Following previous attempts in utilizing LLMs for reasoning and planning [KGR22, HAP22, HXX23, YZY22], we prompt a pre-trained LLM to generate motion failure reasoning and action sequences in text format. Since we want to limit the domain-specific prior provided to the LLM, we use zero-shot prompting [KGR22] without providing any planning examples. We adopt Chain-of-Thought (CoT) prompting [WWS22, YZY22] to have the LLM generate the reasoning and action sequence in an autoregressive manner. The prompt fed into LLM comprises the following contents: (i) a system message that provides the global context to the pre-trained LLM and activates its planning capability, (ii) a task description specifying the environment, goal, and available primitive actions, (iii) the textualized initial environment state, (iv) the trace of motion planning feedback, and (v) the output format. Note that most prompt content will remain unchanged for different planning iterations of the same TAMP problem. Only the motion planning feedback trace in (iv) will be updated as new failed action sequences are added. Fig. 4.2(a) illustrated an example of reasoning and planning with LLM.

We implement two strategies for the pre-trained LLM to generate a new action sequence that improves on the previous one: (i) *backtrack*, where we expect the LLM to backtrack to a previous action that has feasible motion, and continual to generate actions that complete the plan, and (ii) *from scratch*, where we expect the LLM to directly generate a new action sequence that attempts to avoid the motion failure happened to its previous output. We achieve this by designing the system message and output format description in the prompt. We show the prompt template for the two variants in Fig. 4.3, where the content specific to each variant is highlighted.

4.4.3 Synthesizing Motion Planning Feedback

We realize a ground action $\underline{a}$ by calculating a collision-free trajectory τ with a sampling-based motion planner, *e.g*. Bi-directional Rapidly-exploring Random Trees (BiRRT) [LaV06]. The input to the motion planner includes the initial environment state (includes the robot state), and a goal pose of the robot end effector specified by the continuous action parameters of $\underline{a}$. The motion planner samples and searches for a collision-free joint space trajectory for the robot to reach the goal end-effector pose.

By default, the motion planner reports a binary signal that indicates whether there is a feasible trajectory. It does not give more abstract-level feedback, which explains why motion planning fails. As a result, the TAMP planners can acquire useful feedback from motion planning failures to improve high-level planning. To this end, we additionally synthesize semantically meaningful motion-level feedback so that LLM^3 can improve on previous failures more effectively. We observe that typical motion planning failures can be categorized into two types, *i.e*., collisions and unreachability. Therefore, we synthesize categorized motion planning feedback following the templates below (Fig. 4.4):

(A) The goal configuration is in collision with `object`.

(B) The goal configuration has no feasible inverse kinematics (IK) solution.

System message: You are an AI robot that generates a plan of actions to reach the goal... You are expected to correct the plan incrementally (on top of the last plan) to avoid motion failure. This may involve sample new parameters for the failed action or reverse one or more succeeded actions for backtracking... You are expected to generate a plan from scratch.

Task description: A robot arm is tasked to pack boxes into a basket on a table. The robot sits at (0, 0), and faces the positive x-axis, while the positive z-axis points up. The robot is equipped with primitive actions, each taking a list of objects and continuous parameters as input:

- `pick`([`obj`], {}): pick up `obj`, with no parameters.
- `place`([`obj`], {"x" : $[0.0, 1.0]$, "y" : $[-1.0, 1.0]$, "$theta$" : $[-3.14, 3.14]$}): place `obj` at location (x, y) with the planar rotation theta, where x ranges $(0.0, 1.0)$, y ranges $(-1.0, 1.0)$, and $theta$ ranges $(-3.14, 3.14)$.

Initial state: {init_state}

motion planning feedback trace: {trace}

Output format: Please generate output step-by-step, which includes your reasoning for the failure of last plan as well as the generated plan... Answer the questions: (i) what is the cause of the failure of last plan? (ii) can altering action parameters for the failed action solve the problem... (iii) do we need to reverse one or more succeeded actions executed before the failed action... (ii) what is your strategy to generate a new plan from scratch to accomplish the task goal? Please organize the output following the JSON format below: { "Reasoning": "My reasoning for the failure of last plan is ...", "Full Plan": ["pick(['red_box'], {})", "place(['red_box'], {'x': 0.51, 'y': 0.02, 'theta': 0.00})", ...] }

Figure 4.3: **Prompt templates used by LLM^3.** We show alternative contents specific for the *backtrack* variant in orange and *from scratch* variant in blue.

(C) The goal configuration is collision-free and reachable.

In practice, we integrate the motion planner with an additional IK solver and collision checker to obtain these feedback, finding this design to be practically effective.

4.5 Simulations and Experiments

In simulations, we initially perform an ablation study on our LLM^3 framework in two settings of the tabletop box-packing task, quantitatively evaluating its effectiveness based on i) the planning success rate (%SR), ii) the number of LLM calls (#LM), iii) and the number of motion planner calls (#MP). Additionally, we demonstrate the role of LLM as an informed action parameter sampler by comparing it to a baseline utilizing random sampling strategies. Finally, we validate the proposed LLM^3 framework through experimentation on a perception-integrated physical robotic manipulator, confirming its validity in real-world scenarios.

4.5.1 Simulation Setup

We developed a PyBullet-based simulation environment for our box-packing tasks, as illustrated in Fig. 4.5a. In **Setting 1**, three different sets of objects are given with increasing total sizes, while the basket size remains unchanged. This task requires LLM^3 to oversee potential collisions among objects and the robot throughout the action sequence. LLM must reason why collisions occur and

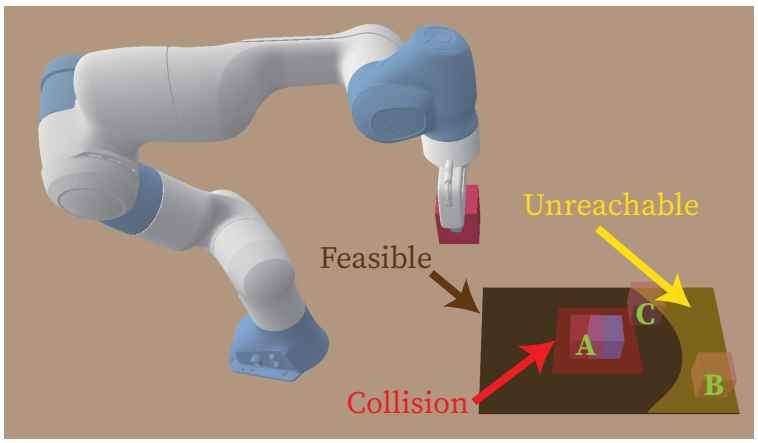


Figure 4.4: **Three types of possible motion planning outcomes.** A: the object placement is in collision with an existing object. B: the object placement is beyond the robot's reach. C: the object placement is feasible.

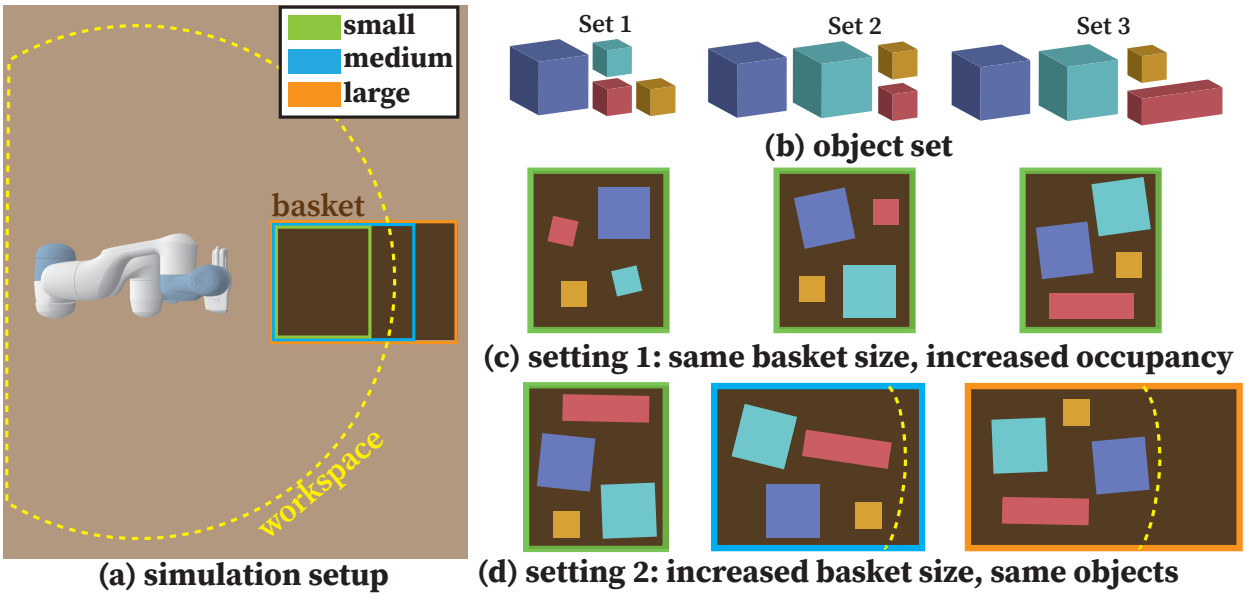


Figure 4.5: **The box-packing task setup in a simulated environment.** (a) The task requires the robot to place one of (b) three sets of objects fully into the basket. (c) In setting 1, the total object size increases but the basket sizes remain the same. All baskets are within reach. (d) In setting 2, the basket size increases, but some portions of baskets are longer within the robot's reach.

adjust previous actions to ensure feasible task and motion plans. **Setting 2** involves placing the Set 3 objects in baskets of increasing sizes. Here, the robot cannot access the entire basket region but encounters a collision likelihood similar to the most crowded condition in Setting 1. In this setting, the task becomes more complex as it challenges LLM to reason about the robot's workspace and adjust the plan accordingly. Throughout the simulations, we utilize GPT-4 Turbo [Ope23a] as the LLM planner and BiRRT [LaV06] as the motion planner, with 10 attempts for each setting.

4.5.2 Ablation Study

The conducted ablation study compares the proposed LLM^3 with baseline methods:

1. **LLM^3 Backtrack**: The proposed LLM^3 framework *backtrack* variant. See Fig. 4.3
2. **Backtrack**: The LLM proposes plans with backtracking but without motion planning feed-

Table 4.1: **Ablation study**

Method	Setting 1									Setting 2								
	Easy			Medium			Hard			Small			Medium			Large		
	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP
LLM^3 Backtrack	**100**	**1.6**	**11.8**	**100**	**4.4**	**28.4**	60	11.4	39.8	60	11.4	39.8	**80**	**9.5**	**50**	50	13.5	44.8
Backtrack	100	1.8	12.6	90	6.3	32	40	15.1	55.3	40	15.1	55.3	30	14.6	16	30	15.8	48
LLM^3 Scratch	100	1.7	13.2	100	7	46.1	**70**	**8.8**	**30.9**	**70**	**8.8**	**30.9**	70	11.5	50.2	**60**	**10.6**	**32**
Scratch	100	2.4	17.6	60	12.3	45.3	50	13.7	42.8	50	13.7	42.8	30	16.2	45.7	40	13.2	24

back (line 7).

3. **LLM^3 Scratch**: The proposed LLM^3 framework *from scratch* variant. It replans the entire action sequence, incorporating motion planning feedback if any action fails. See Fig. 4.3

4. **Scratch**: The LLM plans the action sequence once and executes the plan.

Three evaluation criteria are considered: The number of LLM calls (#LM) counts how many times the LLM API is called during planning. A lower #LM indicates that the planner can produce a feasible task plan more efficiently. In each attempt, #LM has a maximum cap of 20; attempts with over 20 #LMs will be counted as a failure. The total success rate is recorded as %SR. Additionally, the number of motion planner calls, #MP, is another critical criterion as in traditional TAMP approaches, where massive and time-consuming motion planner calls are the main cause for their inefficiency. The study results are summarized in Table 4.1.

In both settings, integrating motion planning feedback results in a decrease in #LM and #MP, along with an increase in %SR for both *backtrack* and *scratch* strategies. This indicates that the LLM can reason about failures from motion planning feedback, and importantly, propose adjusted task plans and action parameters that are more likely to produce feasible motions. Surprisingly, no clear evidence suggests that utilizing backtracking is superior to replanning from scratch. We have observed distinct behaviors among different strategies employed by LLM^3. Specifically, when using backtracking, LLM sometimes consistently adjusts the action parameter of the specific action that *directly* led to failure, without adjusting previous actions. In contrast, the LLM^3 employing a

planning from scratch strategy simply re-samples all actions and parameters, occasionally resulting in a slightly lower #LM.

4.5.3 Action Parameter Selection

This study examines whether an LLM can function as an informed action parameter sampler, using the **Medium** setup from **Setting 1** in the previous section. In this study, the action sequence is predetermined and consists of a total of 8 steps, where the robot sequentially performs pick and place actions to relocate all four blocks into the basket. However, the specific placement location is not provided, requiring the action parameters to be sampled to ensure the feasibility at the motion level. We implement three methods, each runs 50 attempts, for comparison:

1. **Random Samples**: A random sampler is implemented to independently and uniformly sample all the block placement locations within the basket region.
2. **LLM**: The LLM is provided with the task setting and the symbolic action sequence, prompting it to select the action parameters to make the action sequence feasible.
3. **LLM + Feedback**: Building upon 2), the LLM is additionally provided with motion planning feedback if the previous attempt fails.

The results summarized in Table 4.2 demonstrate the efficacy of using LLM as an informed action parameter sampler in the context of box-packing tasks. Specifically, while random sampling

Table 4.2: **Comparison of different sampling strategies**

Method	#Iteration	#MP
Random	109.6	663.1
LLM	10.8	70.2
LLM + Feedback	**7.9**	**53.2**

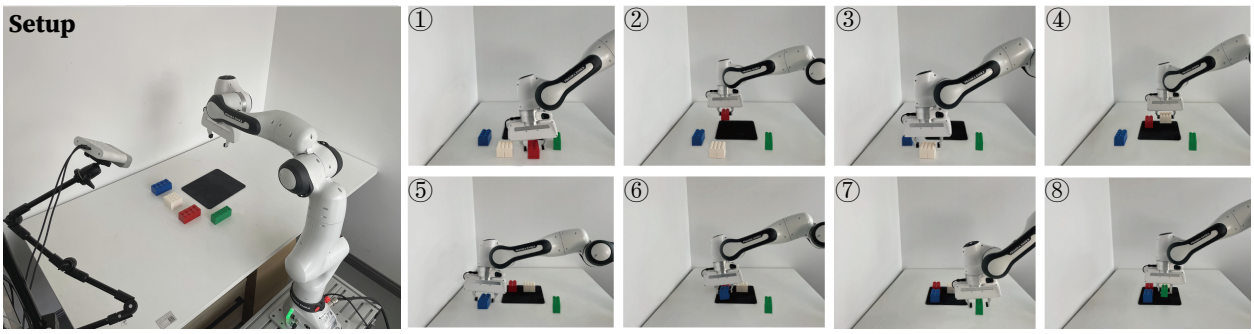


Figure 4.6: **The real-world experiment on a physical robot.** The figure to the left shows the box-packing task setup. Actions ① to ⑧ are proposed by LLM^3 and successfully carried out by the physical manipulator.

requires an average of 109.6 iterations and 663.1 #MP to achieve feasible action sequences, the LLM substantially reduces them to an average of 10.8 iterations and 70.2 #MP. When incorporating motion planning feedback alongside the LLM, the sampling requirement decreases even further to an average of 7.9 iterations and 53.2 #MP. These findings underscore the ability of LLMs to efficiently select action parameters that align with task constraints, resulting in a notable reduction in the number of motion planning calls needed to generate feasible action sequences. Moreover, the additional benefit gained from integrating motion planning feedback highlights the importance of incorporating real-time feedback mechanisms to further refine and optimize the sampling process. Overall, these results highlight the potential of LLMs as a valuable tool to improve the efficiency and effectiveness of robotic manipulation tasks, particularly in scenarios where efficient action parameter selection is crucial for improving planning performance.

4.5.4 Experiment Setup

To validate the effectiveness of our proposed method in a real-world setting, we conducted an experiment using a Franka Research 3 manipulator. The goal was to demonstrate the robot's ability to perceive and manipulate objects in an environment with uncertainties in both perception and execution. The robot observed a single point cloud from a third-person view RGB-D camera, cap-

turing the workspace containing various objects such as blocks and a plate. To identify and locate individual objects, we employed Grounded Segment Anything [RLZ24] for object segmentation, initially segmenting objects in the 2D RGB image and then projecting the results onto the corresponding 3D point cloud. This approach yielded per-object point clouds, essential for planning and executing manipulation tasks.

Fig. 4.6 presents a qualitative evaluation of our method, where the robot was tasked with placing all the blocks on the plate. The results demonstrate that our method enabled the robot to successfully identify and manipulate objects despite uncertainties and challenges in the cluttered environment. The successful execution of this experiment validates the practicality and robustness of our approach, showcasing its potential for various real-world applications such as object sorting, assembly tasks, and household assistance.

4.6 Conclusion

This chapter presents LLM^3, a new TAMP framework powered by the pre-trained LLM. LLM^3 leverages the rich knowledge encoded in and the powerful reasoning capability processed by LLMs to (i) propose action sequences without requiring a prior planning domain, (ii) generate continuous action parameters for the robot motion planner, and more importantly, (iii) refine task and/or motion plans in response to motion planning failures. Following validation through various simulations and experiments, we demonstrated that LLM^3 effectively produces and refines task and motion plans for box-packing problems, exhibiting promising potential in addressing previously unspecified tasks. Our study also reveals that although the pre-trained LLM can generate action parameters more efficiently than random samplers, it still requires multiple feedback iterations and motion planner calls. Looking ahead, the incorporation of in-context learning or fine-tuning techniques holds promise for further enhancing its efficiency, representing a crucial step towards empowering robots to tackle emerging tasks in real-world scenarios.

Part III

Learning: Object and Relation Symbol Learning for Generalizable Planning

CHAPTER 5

Learning Relational Predicates for Generalizable Task Planning

While planning with language model knowledge is promising, it requires manually defining object relations in state descriptions and struggles in long-horizon tasks. In this chapter, we present InterPreT, an LLM-powered framework for robots to learn symbolic predicates (*i.e.*, relational symbols) from language feedback of human non-experts during embodied interaction. The learned predicates facilitate the learning of symbolic operators, *i.e.*, the symbolic world models that capture action preconditions and effects. By compiling the learned predicates and operators into a PDDL domain on-the-fly, InterPreT allows effective long-horizon planning toward arbitrary in-domain goals using a PDDL planner. In both simulated and real-world robot manipulation domains, we demonstrate that by uncovering predicates and operators in simple tasks, InterPreT generalizes strongly to novel tasks with significantly higher complexity. The materials in this chapter have been published in [HZZ24].

5.1 Introduction

Effective long-horizon planning is a long-standing challenge in robotics [ZTB21, XMH19]. Imagine a household robot that prepares a meal in your kitchen. It must be capable of generating faithful multi-step action plans to manipulate novel objects and achieve diverse task goals. Recently, Large Language Models (LLMs) have shown the ability to decompose a high-level task goal into semantically meaningful sub-tasks leveraging the vast amount of world knowledge they encode [LPP22, HAP22]. They exhibit the emergent property of acquiring planning capabilities from a few in-context examples [DLD22, HAP22]. Researchers have successfully applied LLM-

based planners in real-world robotic tasks [ABB22, HXX23, SBM23, LHX23], where they can easily incorporate various forms of feedback and produce plans in novel situations. Nevertheless, LLM-based planners still struggle to generalize strongly to long-horizon tasks, and they offer no performance guarantees [LJZ23, VOS22, SHS22].

In contrast, classical planners [LaV06, Rus10] provide complementary strengths in generating long-horizon plans with formal guarantees. At the heart of these planners are *predicates*, which are binary-valued functions that map environment states to high-level symbolic representations, *e.g.*, a function that transforms the workspace observation into semantic relations such as `on_table(apple)`. With these symbolic predicates, we can subsequently model state transitions with symbolic *operators* [FN71], describing the preconditions and effects of the robot's actions on the symbolic states. The predicates and operators together form a PDDL domain [FL03], allowing a planning algorithm to generate plans for arbitrary in-domain tasks [Hel06]. Despite the wide adoptions of planning algorithms in robotics [KL11, Tou15, GLK20], these methods usually require substantial manual effort and domain expertise to meticulously design the predicates and operators, hindering their applicability to real-world problems.

To combine the best of both worlds, there has been a growing interest in integrating learning methods with planning algorithms. Notable efforts have been made to learn symbolic representations from interaction data using unsupervised learning methods [KKL18b, JRK21, LSA19, SCK23, ASP22]. However, without explicit guidance, they struggle to uncover predicates that capture task-relevant semantic relations to facilitate planning. Meanwhile, cognitive studies [Man92, GHM10] have shown that human infants are remarkably efficient in acquiring new predicate-like relational concepts, such as spatial relations for stacking blocks, from the language feedback of caregivers during physical play. Inspired by these, we envision an *interactive learning* scheme that enables a robot to rapidly learn useful abstractions for planning from online human feedback.

We hypothesize that for robots to achieve human proficiency in learning predicates for planning, they must possess an ability similar to infants to learn from the rich human language feedback in an interactive manner. Recent work has incorporated human language feedback into learning

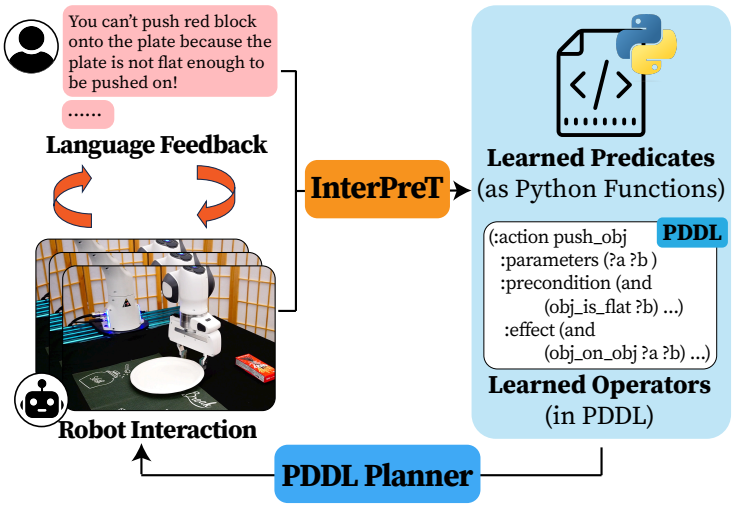


Figure 5.1: **InterPreT learns predicates as Python functions and operators in PDDL from human language feedback during embodied interaction.** The learned predicates and operators can be leveraged by a PDDL planner for planning unseen tasks with more objects and novel goals.

reward functions [MLW23] and motion policies [LCZ23]. The crux of these methods is to harness the capabilities of pretrained LLMs [ZZL23], in particular GPT-4 [Ope23a], in understanding natural language input, performing reasoning [KGR22, YZY22], and generating text-based responses (computer programs [CTJ21], *etc*.). Following this line of work, we present **InterPreT** (**Inter**ative **Pre**dicate Learning for **T**ask Planning), the first framework for robots to learn planning-oriented predicates from interactive language feedback, as depicted in Fig. 5.1. InterPreT formalizes predicate learning as generating Python functions with GPT-4, which are iteratively refined based on human language feedback. These predicates (as Python functions) can access raw environment states with Python perception APIs and freely compose logic structures and arithmetical computations (*e.g*., with NumPy) to form complex semantics. With the learned predicates, we can easily learn symbolic operators from the robot's interaction data using a cluster-and-search algorithm [SCT21]. The learned predicates and operators are compiled into the PDDL format on the fly to be used by a planner. LLMs' capabilities of open-world text processing and symbolic planners' performance guarantees together empower our approach to generalize strongly to arbitrary tasks in

the target domains.

Specifically, we consider language feedback for learning two types of planning-oriented predicates, *i.e.*, goal predicates and action precondition predicates [KKL18b]. These predicates play an essential role in indicating task progress and determining action feasibility, respectively. We design a concise and natural communication protocol to incorporate this feedback:

- **Feedback for learning goal predicates**: At the beginning of each task, the human user specifies the goal, *e.g.*, "put plate on table mat". Then, it signals when the robot achieves the goal and it explains any unsatisfied conditions if the robot mistakenly declares success.
- **Feedback for learning precondition predicates**: The human user verifies the feasibility of the action the robot proposes to execute next. They explain any violated preconditions if the action is infeasible, *e.g.*, "you can't pick up the plate because it is too large for the gripper to grasp", or otherwise confirm that the action is feasible, *e.g.*, "you can go ahead and pick up red block".

This protocol allows InterPreT to verify and refine the learned predicates from time to time, enabling predicate learning with closed-loop feedback.

In the experiments, we evaluate InterPreT's effectiveness in a suite of simulated and real-world robot manipulation domains. These domains are designed such that their dynamics can be modeled using specific predicates and operators, which the robot must uncover. We first have InterPreT learn predicates and operators by having the robot interact with a series of simple training tasks while receiving natural language feedback from human users. We then test the learned predicates and operators on harder tasks involving more objects and novel goals. We show with qualitative and quantitative results that: (i) InterPreT learns valid predicates and operators that capture essential regularities governing each domain. (ii) The learned predicates and operators allow the robot to solve challenging unseen tasks requiring combinatorial generalization, with a 73% success rate in simulation, outperforming all baselines by a large margin. (iii) InterPreT can effectively handle

real-world uncertainty and complexities, operating with considerable performance in real-world robot manipulation tasks.

5.2 Related Work

5.2.1 Learning Symbolic Representations for Planning

Learning symbolic abstractions of complex domains for effective planning is a long-standing pursuit in the planning community [PZK07, JLT13, UP15, KKL18b, ASP22, SCK23, ZHJ23, RJO23]. Previous methods have focused on discovering propositional [KKL18b] or predicate state symbols [JRK21] from embodied experience. These symbols are usually acquired by composing predefined features [PZK07, LSA19, CST22, SCK23], or learning statistical [JLT13] or neural network [ASP22] models with clustering [UP15, KKL18b, JRK21] or representation learning techniques [UAR21, Asa19, ASP22]. Such learning often relies on unsupervised objectives like minimizing state reconstruction error [UAR21, Asa19, ASP22], prediction error [JLT13, UAR21, ASP22], bisimulation distance [CST22] or planning time [SCK23]. However, these approaches struggle to capture high-level semantic relations [Asa19, ASP22] and often require manual feature engineering [LSA19, CST22, SCK23].

Supervised learning has also been explored to ground semantic predicates to continuous observations, *e.g.*, images or continuous states [XZC17, MGK19]. While large-scale annotated datasets [KZG17] are available to learn general-purpose predicate grounding models, fine-tuning with task-specific data is still needed for learned predicates to serve reasoning and planning in specific domains [ZDA23, KHC23, GWZ23]. To reduce annotation needs, prior works have employed active learning [BPY22, LS23] or novel labeling techniques [MB22, MLT22], but a minimum of 500-1000 labels [BPY22, LS23] are still required per predicate.

Our work builds on this line of research in learning symbolic abstractions from interaction data and weak supervision. We mitigate limitations of unsupervised methods by learning from natural

language feedback. Meanwhile, we are able to learn semantic predicates as Python functions from a few data samples, leveraging the code generation capability and world knowledge of GPT-4.

5.2.2 Large Language Models-enabled Planning and Learning

Large Language Models [ZZL23] have shown remarkable abilities in encoding vast semantic knowledge and demonstrate emergent capabilities in learning, reasoning, and planning with few-shot or even zero-shot prompting [BMR20, KGR22, HAP22]. Pretrained LLMs have been applied as planners in text-based environments with natural language instructions and feedback [YZY22, HAP22, LPP22, SCB23, WCL23]. For *grounded* planning in realistic robotic domains, a common approach is to utilize out-of-the-box perception models to convert raw observations into textual descriptions for LLMs to consume [SWW23, WWX23, HXX23, DKD23, WHJ24, ZZH24], or provide perception and action APIs for LLMs to generate executable programs [LHX23, SBM23]. However, these perception models struggle to capture complex task-relevant information like semantic object relations without task-specific tuning [ZDA23, GWZ23]. Leveraging GPT-4's power, our work effectively acquires meaningful task-relevant predicates to facilitate grounded planning.

Pretrained LLMs are also leveraged to enhance robot agent intelligence by generating formatted outputs (*e.g.*, code, formal language) and refining them based on language feedback via iterative prompting. They have been used as interfaces between natural language and robotics modalities like formal planning languages [XYZ23, LJZ23] (*e.g.*, PDDL [FL03]), reward functions [MLW23, YGF23] and trajectories [LCZ23]. Specifically, Voyager [WXJ23] uses GPT-4 to construct an automatic curriculum and a skill library to build lifelong learning agents, while Eureka [MLW23] and OLAF [LCZ23] leverage GPT-4 for learning from language feedback effectively by prompting. Inspired by these works, we learn predicates from language feedback by generating and iteratively refining Python functions with GPT-4.

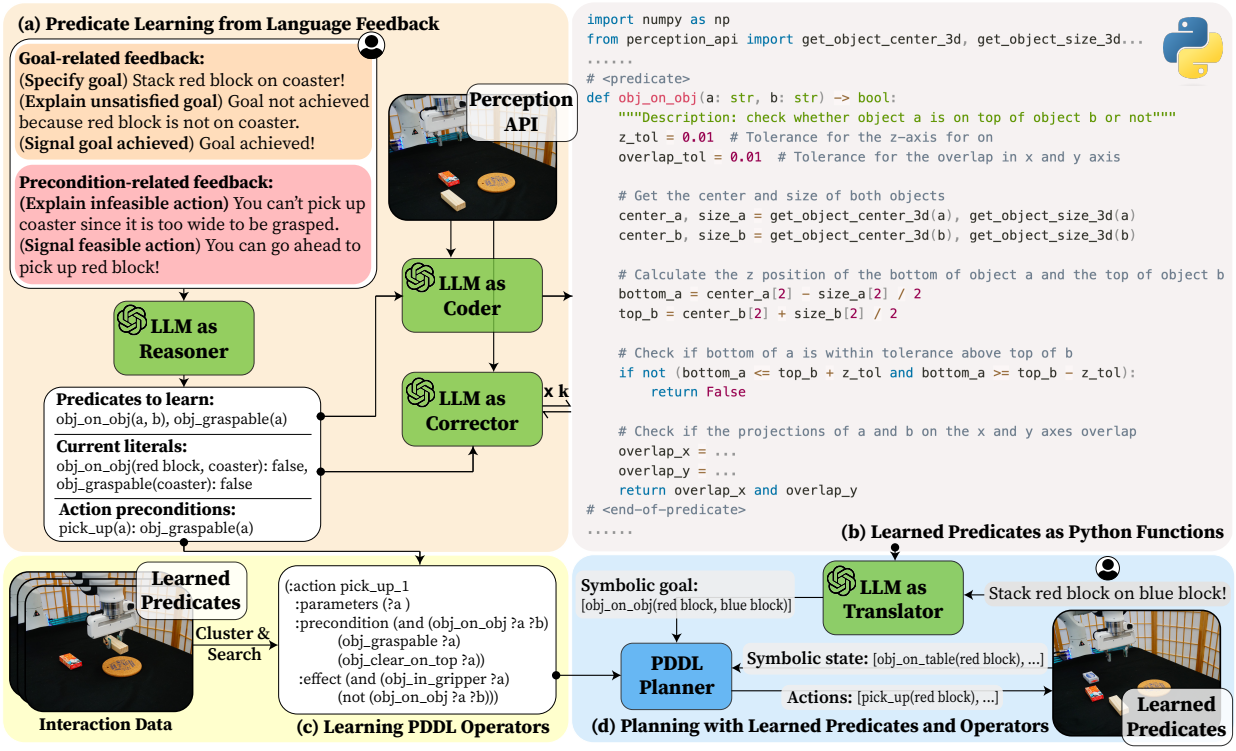


Figure 5.2: **The system architecture of InterPreT.** (a) We design three GPT-4-enabled modules that operate sequentially to identify planning-oriented predicates and generate predicate functions based on language feedback. (b) An example predicate function learned. (c) With the learned predicates, we learn PDDL operators with a cluster-then-search algorithm. (d) The learned predicates and operators enable effective task planning, as we translate language goals into symbolic goals.

5.3 Preliminaries and Problem Setup

We consider robot task planning in a continuous state space O with language goal specifications G. Without losing generality, we assume the states are factorized with respect to a set of objects E, where such information can be obtained using mainstream perception models like object detectors. The robot is equipped with a library of primitive actions A, where each $a \in A$ is parameterized by object variables and can be grounded to certain objects to produce an executable action $\underline{a}$, *e.g.*, `Pick(cup)`. Then, a task planning problem is to find a sequence of actions $\underline{a}_{1:T}$ to reach a final state o_g that satisfies a language goal $g \in G$ from an initial state $o_0 \in O$.

Following the classical planning formulation [Rus10, FN71], we aim to learn predicates Ψ to abstract the state space O into a symbolic one S for effective and generalizable planning. A predicate $\psi :=< d_\psi, f_\psi >\in \Psi$ defines a function f_ψ that captures a symbolic relation among a list of object variables, with its semantic meaning described as d_ψ. The function $f_\psi : O \times E^k \rightarrow \{0, 1\}$ takes a continuous state $o \in O$ and a list of k objects $(e_1, e_2, ..., e_k) \in E^k$ and outputs a binary value indicating whether the relation holds or not. For example, a predicate `on(a,b)` can be applied to check whether `cup` is physically on `plate`, producing a positive literal `on(cup, plate)` or a negative literal $\neg$`on(cup, plate)`. Then the symbolic state s of a continuous state o can be obtained by collecting all positive literals at state o given predicate set Ψ and object set E, denoted $s = \texttt{Parse}(o; \Psi, E)$.

With the object-factorized symbolic state space S, we model the preconditions and effects of primitive actions with symbolic operators Ω. Each symbolic operator $\omega \in \Omega$ corresponding to a primitive action a is characterized by a precondition set CON (literals must hold before executing a), and adding and deleting effect set EFF^+ and EFF^- (literals added and removed from symbolic state s after executing a). These symbolic operators are *lifted* by design, enabling the evaluation of preconditions and effects for any executable version $\underline{a}$ obtained by applying the primitive action a to any objects. With the learned predicates Ψ, we further learn the symbolic operators Ω of all primitive actions A to achieve generalizable task planning. The learned predicates and operators can be compiled into a PDDL domain. By converting a language goal $g \in G$ into a symbolic goal s_g [LJZ23, XYZ23], such a PDDL domain can enable effective planning using an off-the-shelf classical planner [Hel06].

5.4 Method

In this section, we present the InterPreT framework that learns predicates and operators from language feedback for planning. The overall architecture is depicted in Fig. 5.2. There are five essential modules that operate together to empower InterPreT: (i) ***Reasoner***, which analyzes language

feedback to identify new predicates and extract task-relevant information (*e.g.*, predicate labels, action preconditions), (ii) ***Coder***, which generates Python functions to ground the new predicates, (iii) ***Corrector***, which iteratively refines existing predicate functions to align their predictions to the extracted predicate labels, (iv) ***Operator Learner***, which learns operators from interaction data based on the learned predicates, and (v) ***Goal Translator***, which translates language goal specifications into symbolic goals to enable planning. Below, we elaborate on the core GPT-4-powered modules-*Reasoner*, *Coder* and *Corrector*-that enable predicate learning, and briefly introduce the rest, which are mainly adapted from existing works.

Given language feedback l_t at time step t, our objective is to learn new predicates and refine existing predicates Ψ_{t-1}, producing an updated set of predicates Ψ_t. For simplicity of notation, we denote the textual descriptions of predicates as $\{d_\psi\}$ and the corresponding predicate functions as $\{f_\psi\}$ for any predicate set Ψ. We decompose the predicate learning process at time step t into three sequential sub-steps (see Fig. 5.2(a)): (i) *Reasoner* identifies new predicates with descriptions $\{d_{\psi_{new}}\}$ and extracts current state literals that provide predicate labels $\{y\}$, (ii) *Coder* generates new predicate functions $\{f_{\psi_{new}}\}$, and (iii) *Corrector* refines existing predicate functions to fix execution errors and match their predictions to $\{y\}$. Formally, we summarize this process in Eq. (5.1):

$$\begin{aligned}
\text{Reasoner}: \{d_{\psi_{new}}\}, \{y\} &= f_{Reason}(l_t, \{d_{\psi_{t-1}}\}), \\
\{d_{\psi_t}\} &= \{d_{\psi_{new}}\} \cup \{d_{\psi_{t-1}}\} \\
\text{Coder}: \{f_{\psi_{new}}\} &= f_{Code}(\{d_{\psi_{new}}\}, \Psi_{t-1}), \\
\{f_{\hat{\psi}_t}\} &= \{f_{\psi_{new}}\} \cup \{f_{\psi_{t-1}}\}, \\
\text{Corrector}: \{f_{\psi_t}\} &= f_{Correct}(o_t, \{y\}, \{f_{\hat{\psi}_t}\}), \\
\Psi_t &= \{< d_{\psi_t}, f_{\psi_t} >\},
\end{aligned} \tag{5.1}$$

where f_{Reason}, f_{Code}, $f_{Correct}$ are parameterized by GPT-4 with varying prompt templates, and the initial predicate set is empty, *i.e.*, $\Psi_0 = \varnothing$. Note that we omit some of the output terms irrelevant to predicate learning for clarity.

5.4.1 Reasoner

The *Reasoner* module is designed to identify essential predicates and extract task-relevant information from goal-related or precondition-related language feedback; see the top left corner of Fig. 5.2(a) for examples of these feedback types. We tailor *Reasoner* to each language feedback type using different prompt templates, as detailed below. We highlight the language feedback in blue, and the GPT-4 output in orange. We employ CoT prompting [WWS22] for *Reasoner* to provide the complete reasoning trace and in-context learning [KGR22] to enable *Reasoner* to learn from a single example. These techniques are applied to all LLM promptings to facilitate robust response generation.

5.4.1.1 Goal-related feedback

- **(Specify task goal)** Given a natural language goal, *Reasoner* identifies new goal predicates and converts the language goal into symbolic form using existing and new predicates:

  Context: ...{example} {objects} {existing_predicates}...

  Goal: Stack red block on coaster.

  Reasoning: The goal can be captured by a symbolic literal obj_on_obj(red block, coaster). As predicate obj_on_obj(a, b) is unknown, we need to learn it.

  Predicates to learn: {"obj_on_obj(a, b)": "check whether object a is on object b"}

  Symbolic goal: {"obj_on_obj(red block, coaster)": true}

  The identified predicates to learn $\{d_{new}\}$ are feed into *Coder* for predicate function generation, and the converted symbolic goal s_g is recorded for robot planning and exploration.

- **(Explain unsatisfied goal)** Given explanations for unsatisfied goal literals, *Reasoner* extracts the current symbolic literals from language feedback, which provide predicate labels $\{y\}$. These predicate labels are crucial for *Corrector* to correct the generated goal predicates.

Context: ...{example} {objects} {existing_predicates}...

Human explanation: You haven't reached the goal because red block is not on coaster.

Current symbolic state: {"obj_on_obj(red block, coaster)": false}

- **(Signal goal achieved)** When receiving goal-achieved signals, we use the extracted symbolic goal as the current literals, which provide positive labels $\{y\}$ to supervise goal predicates.

5.4.1.2 Precondition-related feedback

- **(Explain infeasible action)** Given an explanation of an infeasible action, *Reasoner* identifies new precondition predicates to learn $\{d_{new}\}$, reasons about the current symbolic state $\{y\}$, and summarizes the reflected action preconditions CON_{new}, as shown in the example below. The summarized action preconditions only take arguments that exist in those of the action, *e.g.*, for action `pick_up(a)`, we only summarize the preconditions that take no argument or object `a` as an argument. We aggregate CON_{new} into the precondition set CON_a of the corresponding action a, which is maintained for operator learning (described in Section 5.4.4). Additionally, $\{d_{new}\}$ is fed into *Coder* to generate precondition predicate functions, and the predicate labels $\{y\}$ are provided to *Corrector* for correction.

Context: ...{example} {objects} {existing_predicates}...

Infeasible action: pick_up(coaster)

Human explanation: You can't pick up coaster it is too large to be grasped.

Reasoning: ... The precondition of pick_up(coaster) is that it is small enough to be grasped by the gripper... We learn predicate obj_graspable(a) to check whether object a can be grasped by the gripper...

Predicates to learn: {"obj_graspable(a)": "check whether object a is small enough to be grasped by the gripper..."}

New action preconditions: {"action": "pick_up(a)", "new preconditions": {"obj_graspable(a)": true}}

Current symbolic state: {"obj_graspable(coaster)": false}

- **(Signal feasible action)** When an action is signaled as feasible, we confirm that all preconditions in CON_a are satisfied. The labels y for these precondition predicates are obtained and provided to *Corrector* to correct precondition predicate functions.

5.4.2 Coder

Once *Reasoner* identifies a set of new predicates with text descriptions $\{d_{new}\}$, the next step is to construct the corresponding predicate functions $\{f_{new}\}$ to truly learn them. Inspired by recent successes in using pretrained LLMs to generate computer programs for robotic tasks [LHX23, SBM23, WXJ23, MLW23], we design *Coder* to generate these predicate functions as Python code based solely on $\{d_{new}\}$, leveraging the power of GPT-4. We assume the availability of a library of perception API functions that provide access to continuous states, such as the bounding boxes and categories of detected objects. The predicate functions can then be constructed by composing these API functions with classical logic structures and arithmetical computations (*e.g.*, using NumPy), exploiting the flexibility of Python programming. Representing predicates as Python functions offers several advantages: (i) They are semantically rich and interpretable compared to neural

networks [ASP22, UAR21], and have better representation power and more versatile syntax than logical programs [SCK23, CST22]. (ii) They enable one-shot generation purely from the text description without labeled data, leveraging the extensive commonsense priors in GPT-4.

To facilitate the construction of predicate functions, we provide *Coder* with the following primitives: (i) perception API functions for accessing environment states, (ii) the NumPy library for arithmetic computations, and (iii) if-else and loop statements for controlling the logic structure. We also allow *Coder* to create additional utility functions that can be reused to define different predicate functions. This divide-and-conquer strategy helps mitigate the complexity of building predicate functions from scratch. In practice, we prompt GPT-4 with a code snippet demonstrating the usage of primitives by a few examples of utility functions and one example predicate function. Detailed comments are included in these examples to enable CoT prompting. Due to space limitations, we show a partial prompt with an example utility function `get_object_x_range` below, and an example predicate function in Fig. 5.2(b).

```
import numpy as np
from perception_api import get_object_center_3d, get_object_size_3d...

# <utility>
def get_object_x_range(a) -> np.ndarray:
    """
    Get the range of object a along x axis
    :param a: string, name of detected object
    """
    center = get_object_center_3d(a)
    extent = get_object_size_3d(a)
    return np.array([center[0] - extent[0] / 2, center[0] + extent[0] / 2])
# <end-of-utility>

......
```

5.4.3 Corrector

The predicate functions generated by *Coder* are pruned to two types of errors: (i) **execution errors** due to invalid operations or syntax errors in the function, and (ii) **alignment errors** that arise when GPT-4's understanding of predicates misaligns with that of the human user. Inspired by [WXJ23, MLW23], we introduce *Corrector*, which adopts an iterative prompting mechanism to correct the functions and rectify these errors.

When an execution error occurs while calling a predicate function, *Corrector* takes the execution trace from the code interpreter and the erroneous code as input. It then performs zero-shot reasoning on the error and generates the corrected code as output. This process is repeated until the error is resolved or the maximum number of iterations is reached. Similarly, to address alignment errors, *Corrector* is provided with the predicate labels $\{y\}$ and the current predicate functions. It then iteratively refines the functions to align their outputs with the given labels. An example prompt illustrating this process is shown below:

Context: ...{instructions} <code> {observation}...

Alignment error: obj_on_obj(red block, blue block) should be true, but your prediction is false.

Reasoning: This is because we assume object a should be within object b along the x-axis when we define the predicate function obj_on_obj(a, b), which is wrong; in fact, the relation is established when object a is overlapped with b, so we need to correct the function accordingly.

Corrected code: <corrected_code>

5.4.4 Other Components

Given the learned predicates, we implement a variant of the cluster-then-search algorithm [SCT21] to learn operators. This algorithm effectively learns symbolic operators that best capture the action effects and preserve a minimal set of necessary preconditions from a small number of successful and failed interactions. We also incorporate the action preconditions summarized by *Reasoner* into the learned operators. To ensure learning from language feedback with no delays, we run the operator learning algorithm at each interaction step, maintaining an operator set compatible with the up-to-date predicates and interaction experience.

During the training phase, InterPreT learns predicates and operators as the robot interacts with the environment to perform a series of training tasks (detailed in Section 5.5.1.3). We employ a strategy where the robot plans with a classical planner [Hel06] based on the learned predicates and operators 50% of the time, and randomly takes a symbolically feasible action according to the recorded action preconditions otherwise. Empirically, this approach enables a balance between exploration and exploitation.

At test time, we introduce an LLM-based goal translator to convert language goals into symbolic form, following previous works [XYZ23, LJZ23]. We refer the reader to the original papers for a detailed explanation of how the method works. In practice, we find that the GPT-4-based goal translator performs robustly when provided with a few examples.

5.5 Experiments

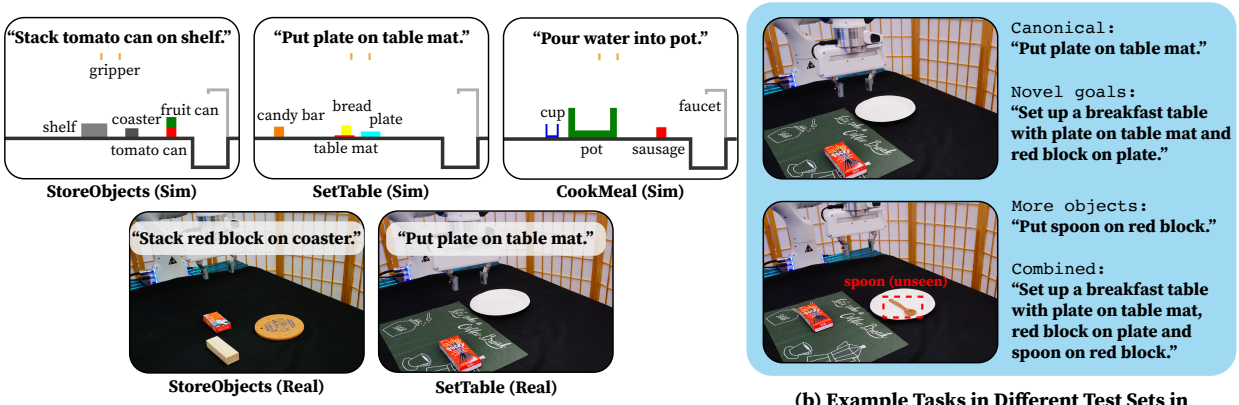


Figure 5.3: **Simulated and real-world domains used in the experiments.** We show example training tasks of all five domains in (a) and demonstrate the design of the 4 test sets in the real-world SetTable domain in (b). In `More objects` and `Combined`, an unseen object "spoon" introduces additional generalization challenges.

We conduct experiments to answer the following questions: (i) Can InterPreT learn meaningful task-relevant predicates and operators from language feedback? (ii) How well do the learned predicates and operators (*i.e.*, PDDL domains) generalize to tasks that involve more objects and novel goals? (iii) Can InterPreT handle perception and execution uncertainties in the real world?

5.5.1 Experimental Setup

We quantitatively and qualitatively evaluate InterPreT on a suite of robot manipulation domains in a simulated 2D kitchen environment [WGK18] and a real-world environment. The domain design, baseline methods, and evaluation protocol are described below.

5.5.1.1 Domain design

We design three simulated domains based on the Kitchen2D environment [WGK18] and two real-world domains that represent the counterpart of the simulation. Each domain is associated with a set of simple and complex tasks, and designed with a ground-truth PDDL domain file specifying the essential symbolic constraints and regularities. The five domains are each demonstrated with an example simple task in Fig. 5.3(a).

- **StoreObjects (Sim and Real)**: This domain involves storing objects on a large receptacle by picking, placing, and stacking actions. It features predicates and corresponding constraints similar to those in the BlockWorld domain [GN91], such as `on(a,b)`, and `on_table(a)`.
- **SetTable (Sim and Real)**: This domain involves rearranging objects to set up a breakfast table. Compared to StoreObjects, it additionally introduces a push action to move large objects (*e.g.*, plates) that cannot be grasped. It features precondition predicates such as `is_graspable(a)` and `is_flat(a)`.
- **CookMeal (Sim only)**: This domain involves putting ingredients into a pot and filling the pot/cups with water. It requires understanding action semantics, featuring predicates such as `is_container(a)`, `in(a,b)` and `has_water(a)`. It also imposes constraints such as the only way to fill a large container (*e.g.*, a pot) with water is by using a cup.

5.5.1.2 Baselines

As there are no prior methods that learn predicates from human language feedback for planning, we compare InterPreT with state-of-the-art LLM-based planners. (i) **Inner Monologue (IM)**[HXX23] generates action plans based on textualized environment states using an LLM. (ii) **Code-as-Policies (CaP)**[LHX23] employs an LLM to generate policy code that invokes perception and action APIs. We also implement variants of IM that incorporate predicates and operators learned with InterPreT. For a fair comparison, all baselines access the environment state through

perception APIs and learn from in-context examples.

- **IM + Object** [HXX23]: A naive IM variant that utilizes the textualized output of perception APIs, *e.g.*, detected objects with positions and categories, as the environment state.
- **IM + Object + Scene** [HXX23]: An IM variant that uses environment states augmented by scene descriptions, obtained using predicates learned by InterPreT.
- **IM + Object + Scene + Precond** [HXX23]: An IM variant that uses the operators learned with InterPreT to check the precondition of actions proposed by IM. Infeasible actions are prompted back to the LLM for replanning.
- **CaP** [LHX23]: A strong CaP baseline that performs precondition checks using "assertion" or if-else statements (akin to ProgPrompt[SBM23]) and hierarchically composes policies for long-horizon planning. We have it generate predicate functions for precondition checks.

5.5.1.3 Evaluation protocol

We adopt a train-then-test evaluation workflow for all domains. For each domain, the robot first learns from a series of 10 simple training tasks accompanied by language feedback. For testing, we design four sets of tasks (10 tasks per set) that pose different levels of challenge to the generalizability of the methods. We present example tasks in different test sets of the real-world SetTable domain in Fig. 5.3(b).

- `Canonical`: Simple tasks with objects and goals seen in training but with different initial configurations.
- `More objects`: Simple tasks with seen goals but involve additional unseen objects.
- `Novel goals`: Complex tasks with seen objects but novel goals that compose goals seen in training tasks.

- `Combined`: Complex tasks with unseen objects and goals, combining the last two setups.

We evaluate the performance of all methods using the success rate on the 10 tasks of each test set. In simulation, we conduct systematic evaluations by running the whole training-testing pipeline 3 times with varied seeds. We directly terminate the episode upon action failure for all methods.

Domain	**Goal Predicates**	**Precondition Predicates**
StoreObjects	`obj_on_obj(a, b)`, `obj_on_table(a)`	`obj_graspable(a)`, `obj_clear(a)`, `gripper_empty()`
SetTable	`obj_on_obj(a, b)`, `obj_on_table(a)`	`obj_graspable(a)`, `obj_clear(a)`, `gripper_empty()`, `obj_is_plate(a)`, `obj_thin_enough(a)`
CookMeal	`obj_inside_obj(a, b)`, `obj_on_table(a)`, `obj_filled_with_water(a)`	`obj_graspable(a)`, `obj_clear(a)`, `gripper_empty()`, `obj_is_plate(a)`, `obj_thin_enough(a)`, `obj_large_enough(a)`, `obj_is_food(a)`, `obj_is_container(a)`

Table 5.1: **Learned goal and precondition predicates in simulated domains.** We report the union of the three runs. While we learn both the positive predicate and its negated counterpart, we only show the positive ones here for clarity. We adjust some of the predicate names to unify them across domains and runs for better readability.

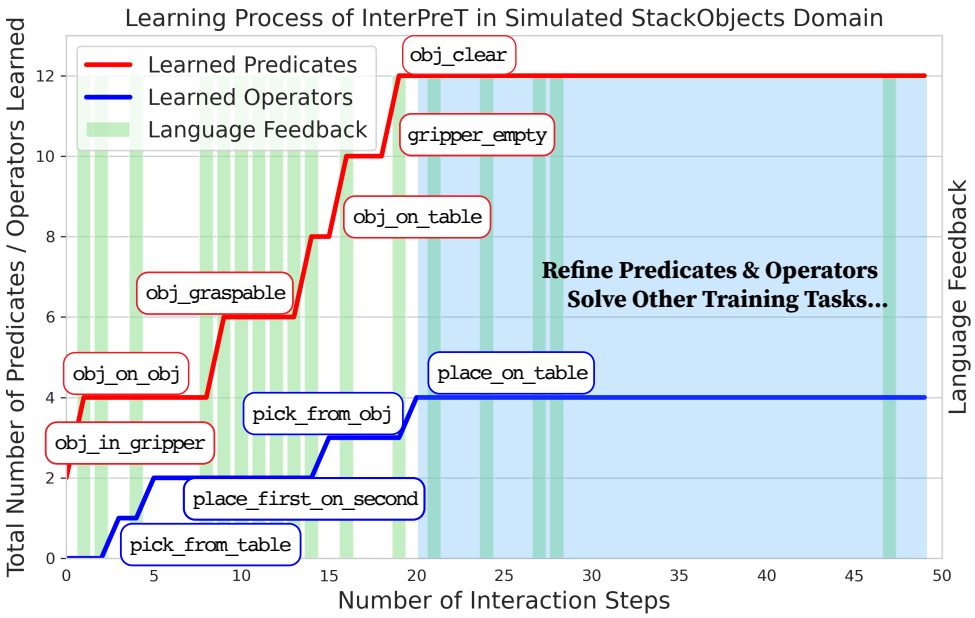


Figure 5.4: **Visualization of one training run in simulated StoreObjects domain.** The total number of learned predicates increases by 2 for each labeled predicate as we also learn its negation. We provide the predicate function of `obj_in_gripper` as an in-context example at Step 0. We empirically label the learned operators with semantic names based on their interpreted meanings.

5.5.2 Experimental results

5.5.2.1 Qualitative analysis

We answer Question (i) by qualitatively analyzing the predicates and operators learned by InterPreT in the simulated domains. Table 5.1 shows InterPreT can effectively learn language-grounded and semantically meaningful goal and precondition predicates in all three domains. We report the union of learned predicates over three runs; we observe that the learned predicates are generally consistent across the runs. Specifically, InterPreT successfully learns goal predicates that acquire the desired task outcomes, such as "fruit can on shelf" and "plate on table" in StoreObjects and SetTable domains and "sausage in pot" and "cup is filled" in CookMeal domain. The learned precondition predicates acutely capture the essential task constraints, such as "fruit can can only be picked up when there is nothing on its top", and "water can only be poured into a container". These

Domain	Test Set	IM + Object [HXX23]	IM + Object + Scene [HXX23]	IM + Object + Scene + Precond [HXX23]	CaP [LHX23]	**InterPreT (Ours)**
StoreObjects	`Canonical`	0.60 ± 0.00	0.90 ± 0.00	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.93 ± 0.12
	`More objects`	0.30 ± 0.00	0.83 ± 0.06	$\mathbf{1.00 \pm 0.00}$	0.83 ± 0.15	0.90 ± 0.17
	`Novel goals`	0.00 ± 0.00	0.87 ± 0.15	0.97 ± 0.06	0.53 ± 0.21	$\mathbf{1.00 \pm 0.00}$
	`Combined`	0.00 ± 0.00	0.77 ± 0.06	0.87 ± 0.15	0.03 ± 0.06	$\mathbf{1.00 \pm 0.00}$
SetTable	`Canonical`	0.80 ± 0.10	0.80 ± 0.10	$\mathbf{1.00 \pm 0.00}$	0.87 ± 0.06	$\mathbf{1.00 \pm 0.00}$
	`More objects`	0.73 ± 0.06	0.83 ± 0.12	$\mathbf{1.00 \pm 0.00}$	0.73 ± 0.15	$\mathbf{1.00 \pm 0.00}$
	`Novel goals`	0.00 ± 0.00	0.10 ± 0.10	0.53 ± 0.33	$\mathbf{0.77 \pm 0.25}$	0.53 ± 0.41
	`Combined`	0.00 ± 0.00	0.03 ± 0.05	0.20 ± 0.16	0.33 ± 0.15	$\mathbf{0.37 \pm 0.45}$
CookMeal	`Canonical`	0.90 ± 0.00	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.06	0.97 ± 0.06
	`More objects`	1.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.93 ± 0.06	$\mathbf{1.00 \pm 0.00}$
	`Novel goals`	0.97 ± 0.06	0.93 ± 0.06	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
	`Combined`	0.00 ± 0.00	0.23 ± 0.15	$\mathbf{0.97 \pm 0.06}$	0.77 ± 0.12	0.83 ± 0.12
Average success rate over `Combined`		0.00	0.34	0.68	0.38	**0.73**

Table 5.2: **Systematic evaluations of the methods on all test sets in simulated domains.** We highlight our method in deep gray and baselines that benefit from our learned predicates and/or operators in light grey. InterPreT achieves a 73% success rate in the most challenging `Combined` test set, outperforming all baselines by a large margin.

well-learned predicates necessarily build the foundations for learning good operators.

We conduct a case study on one training run in the StoreObjects domain. Fig. 5.4 visualizes the process of learning new predicates and operators (represented as red and blue lines, respectively) while provided with intermittent language feedback (indicated by light green bars). Note that the less important feedback for predicate learning, *i.e.*, signaling task success or feasible action, is omitted from the figure for clarity. We observe that InterPreT is able to explore effectively and acquire all predicates in 20 steps of interaction. Based on the predicates learned, InterPreT sequentially learns four operators that exhibit clear semantic meaning. Notably, it recovers two operators `pick_from_table` and `place_on_table` for the same primitive action `place_up` that is executed in different contexts. As the robot blindly explores the domain with inadequate knowledge and continuously proposes infeasible actions, dense language feedback is provided to

explain precondition violations in Steps 8-20. Once InterPreT captures all action preconditions, the robot can freely navigate the environment without human intervention. Fig. 5.4 shows that all predicates and operators are properly initialized at Step 20 and are corrected and refined in subsequent interactions.

5.5.2.2 Evaluating planning and generalization

We systematically evaluate the planning performance of all methods on the four test sets for each simulated domain. Table 5.2 presents the full results, demonstrating the strong generalizability of InterPreT when planning with a classical planner [Hel06]. Note that several baselines utilize predicates and operators learned with InterPreT; their results are shown in light gray, while InterPreT's results are in dark gray. InterPreT achieves success rates over 90% on most test sets. On the challenging `Combined` test set, which requires strong compositional generalizability, it attains an average success rate of 73%, substantially outperforming IM variants (IM + Object, IM + Object + Scene, and IM + Object + Scene + Precond) by 73%, 39%, and 5%, respectively, and the CaP baseline by 35%.

We find the predicates and operators learned with InterPreT enable significantly improved generalization in planning, by providing meaningful relational abstractions and explicit transition modeling. The naive IM variant (IM + Object) struggles to generalize with only textualized state descriptions, solving 0% of `Combined` tasks. However, augmenting states with predicates learned by InterPreT (IM + Object + Scene) boosts the success rate on `Combined` tasks from 0% to 34%. Further ensuring action validity using learned operators (IM + Object + Scene + Precond) rivals InterPreT at 68% average success. This hybrid approach benefits from combining world knowledge in the LLM with validity guarantees from operators. However, we observe that it sometimes fails to reach the goal within the maximum number of steps due to frequent replanning. In contrast, InterPreT perform explicit PDDL planning with learned predicates and operators, and thus can generate optimal long-sequence plans with guarantee.

	From scratch	Bootstrapped
`Canonical`	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
`More objects`	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
`Novel goals`	0.53 ± 0.41	$\mathbf{1.00 \pm 0.00}$
`Combined`	0.37 ± 0.45	$\mathbf{1.00 \pm 0.00}$

Table 5.3: **Bootstrapping predicate learning from previously learned predicates.** Reusing predicates learned in StoreObjects leads to near-perfect predicate learning in SetTable. The transfer of predicates is natural as all predicate functions utilize the same Perception API functions.

We demonstrate the importance of learning from language feedback by comparing InterPreT with CaP, a baseline that generates predicate functions for precondition checks and composes policies for long-horizon planning, but without leveraging language feedback. Although CaP can generate policy code with correct logic based on in-context examples, it occasionally fails to generate accurate predicate functions due to the lack of language supervision. This limitation becomes evident in CaP's poor performance on `Combined` tasks in the StoreObjects and SetTable domains, which require precise predicate understanding for successful long-horizon planning. The superior performance of InterPreT in these challenging scenarios highlights the significant benefits of incorporating natural language supervision compared to CaP.

Furthermore, we explore the transferability of learned predicates to new domains. We investigate the unsatisfactory performance of InterPreT in the SetTable domain, and find that simultaneously learning predicates related to pick-and-place and push actions poses a significant challenge. To address this issue, we bootstrap the learning process with predicates acquired from simpler domains. Table 5.3 demonstrates that initializing InterPreT with predicates learned in the StoreObjects domain leads to near-perfect learning in the SetTable domain, achieving 100% success across all test sets. This finding highlights the potential for reusing previously learned predicates to enhance learning efficiency and planning performance in complex domains.

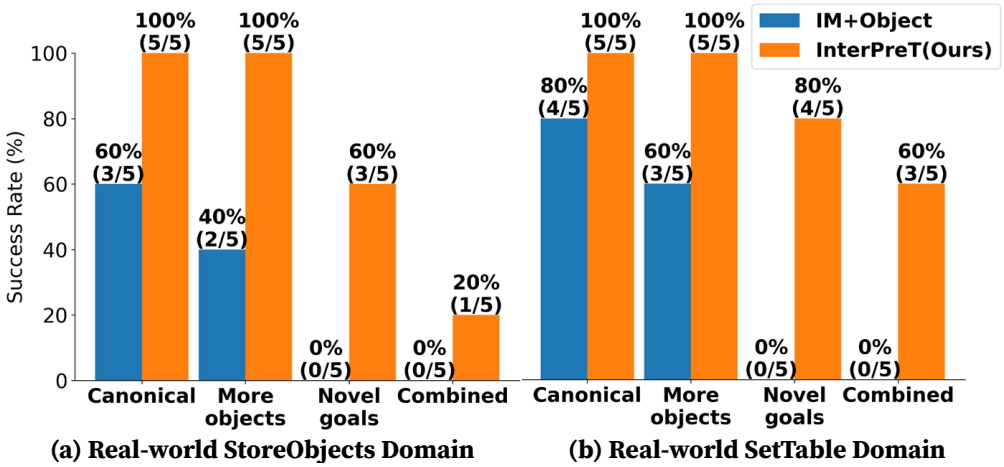


Figure 5.5: **Real-robot evaluations in real-world StoreObjects and SetTable domains.** We train InterPreT once on 10 tasks and test on 5 tasks per test set. Note that the predicate learning in SetTable is bootstrapped from predicates learned in StoreObjects.

5.5.2.3 Real-robot results

We evaluate InterPreT in the real-world StoreObjects and SetTable domains, compared to the vanilla IM + Object baseline [HXX23]. We train InterPreT on 10 training tasks while a human user provides language feedback with a keyboard. We then test all methods on 5 test tasks per test set, with the success rates shown in Fig. 5.5. In the SetTable domain, we directly bootstrap the learning with predicates learned from StoreObjects, as the simulated results have already demonstrated the difficulty of learning from scratch in SetTable. The results indicate that InterPreT can effectively capture symbolic constraints and regularities in real-world settings where perception and execution uncertainties present. In contrast, the baseline struggles to generalize to novel task goals, highlighting the importance of the learned predicates and operators. We observe a severe performance drop for InterPreT in the StoreObjects domain under the `Combined` and `Novel goals` settings. We find that this is attributed to the increased occurrence of primitive action execution failures as the task horizon extends. Despite this, InterPreT still outperforms the baseline by a significant margin, achieving success rates of 60% and 20% in the `Novel goals` and `Combined` settings, respectively.

5.5.3 Additional Analysis and Discussions

Stage		Run time / Iteration		
		Median	Min	Max
Training	Learn predicates	2.94 s	1.05 s	23.32 s
	Learn operators	32 ms	1 ms	97 ms
Testing	Translate goal	1.60 s	0.91 s	5.53 s
	Plan with PDDL	99 ms	75 ms	130 ms

Table 5.4: **Run time breakdown of InterPreT at different training and testing stages.** We show the median, minimum, and maximum values as the statistics are not normally distributed.

Domain	**#LLM Calls**	**#Transitions**	**#Feedback**
StoreObjects	22/31/23	54/75/90	17/26/18
SetTable	38/38/62	41/39/67	31/30/55
CookMeal	32/29/46	62/34/48	23/22/38

Table 5.5: Number of LLM calls (**#LLM Calls**), state transitions collected (**#Transitions**), and language feedback provided (**#Feedback**) across three runs in each domain.

5.5.3.1 Runtime Analysis

To gain insights into the computational efficiency of InterPreT, we measure its runtime in the simulated domains and provide a breakdown by stage in Table 5.4. Due to the variability in runtime across different iterations, we report the median, minimum, and maximum values for a comprehensive overview. The results reveal that the GPT-4-powered predicate learning and goal translation stages constitute the primary computational bottleneck. This is expected as calling the GPT-4 API involves a relatively long waiting time, which is also significantly influenced by the quality of the

Internet connection. However, we anticipate that response time will cease to be a limiting factor for LLMs in the near future, given the rapid advancements in the field.

We also present other relevant statistics in Table 5.5, including the number of LLM calls, successful state transitions collected, and the amount of human feedback provided across three training runs for each domain. While these values exhibit considerable variation due to the inherent randomness in exploration and LLM outputs, InterPreT demonstrates the ability to recover a PDDL domain from a relatively small number of language feedback and interaction data. This highlights the sample efficiency of our approach, which is crucial for practical applications where extensive human feedback and interaction may be costly or time-consuming to obtain.

	Run1	**Run2**	**Run3**
Goal Predicates	`obj_on_obj(a, b)`, `obj_on_table(a)`	`obj_on_obj(a, b)`, `obj_on_table(a)`	`obj_on_obj(a, b)`, `obj_on_table(a)`
Precondition Predicates	`obj_small_enough(a)`, `obj_clear(a)`, `gripper_empty()`	`obj_size_ok_for_gripper(a)`, `no_obj_on_top(a)`, `hand_empty()`	`obj_small_enough_for_gripper(a)`, `obj_free_of_objects(a)`, `gripper_empty()`

Table 5.6: **Learned predicates across three training runs with varied language feedback for the simulated StoreObjects domain.**

	Run1	Run2	Run3
`Canonical`	1.00	1.00	1.00
`More objects`	1.00	1.00	1.00
`Novel goals`	1.00	1.00	1.00
`Combined`	1.00	1.00	1.00

Table 5.7: **Evaluating InterPreT trained with varied language feedback for the simulated StoreObjects domain.** We report the results of all three training runs.

5.5.3.2 Robustness to varied language feedback

Natural language feedback from non-expert human users can be varied, with the same predicate being referred to in different ways. We evaluate the robustness of InterPreT to such varied feedback in the simulated StoreObjects domain by synthesizing diverse feedback templates. Using ChatGPT [Ope], we generate 3 alternatives for each possible feedback, which are randomly sampled during each training step. We conduct three training runs with this varied feedback and perform both qualitative and quantitative evaluations.

Table 5.6 presents the predicates learned across the three runs, demonstrating that InterPreT robustly captures the essential goal and precondition predicates. As the goal specifications are generally consistent, InterPreT learns goal predicates with the same names in all runs. Despite the varied explanations of precondition violations, InterPreT learns precondition predicates with different names but consistent semantics. Table 5.7 shows that the predicates and operators learned from the varied feedback yield robust planning in all test sets. These results demonstrate InterPreT's robustness to diverse language feedback, highlighting its ability to capture the underlying semantics despite variations in the feedback provided.

5.6 Conclusion

We present InterPreT, an interactive framework that enables robots to learn symbolic predicates and operators from language feedback during embodied interaction. InterPreT learns predicates as Python functions leveraging the capabilities of LLMs like GPT-4. It allows iterative correction of these learned predicate functions based on human feedback to capture the core knowledge for planning. The predicates and operators learned by InterPreT can be compiled on the fly as a PDDL domain, which enables effective task planning with a formal guarantee with a PDDL planner. Our results demonstrate that InterPreT can effectively acquire meaningful planning-oriented predicates, which allows learning operators to generalize to novel test tasks. In simulated domains, it achieves a 73% success rate on the most challenging test set that requires generalizability to more objects

and novel task goals. We also show InterPreT can be applied in real-robot tasks. These findings validate our hypothesis that human-like planning proficiency requires interactive learning from rich language input, akin to infant development.

Limitations While showing promise, InterPreT has several limitations that we would like to acknowledge. First, the generalizable planning capability of InterPreT is realized by the learned symbolic operators. This introduces the assumption that the underlying domain can be well modeled at a symbolic level. This is generally not exact, as the physical world is inherently continuous. A promising future direction is to extend InterPreT into the setup of TAMP [GCH21], which considers both symbolic understanding and continuous interactions. Also, the operators learned by InterPreT are deterministic, which falls short of capturing the uncertainty in state transitions. This issue can be mitigated by learning operators with probabilistic effects [KKL18b, ASP22].

CHAPTER 6

Learning Object Symbols for Generalizable Object Cutting

In previous chapters, we majorly studied manipulation tasks in which an object can be treated as a whole and abstracted with its semantic category. However, this assumption does not hold for tasks that involve object fragmentation. In this chapter, we present a novel approach that learns object symbols and a stochastic grammar-based world model for object cutting. We devise a probabilistic framework to learn this grammar from human demonstrations. Then we formulate planning for object cutting as posterior inference, which can be efficiently solved via Monte Carlo Tree Search (MCTS). We show with simulations and real-robot experiments that our approach can generalize to novel setups thanks to the compositionality of the grammar model. The materials in this chapter have been published in [ZHJ23].

6.1 Introduction

Representing object states and understanding how they change with actions are fundamental for robots to manipulate the physical world. In the literature, the primary focus is restricted to *rigid* objects whose states can only be altered spatially, represented with reconstructed 3D geometry [HZJ21, HZJ22, AXW21], estimated 6D poses [DXM20, WB21], semantic keypoints [VRS21, MGF22], or extracted appearance features [DAD18, VCC20]. Recently, *articulated* object understanding in terms of kinematics estimation [JHZ22, JLC21] and part-level object modeling [MZC19, WWZ21, ZZW23], and *deformable* object understanding empowered by physics-based simulation [HLS19, LHL22] further expand a robot's manipulation capabilities towards handling drastic appearance and geometry changes of objects. However, a rigid, articulated, or deformable object

can be treated as a single whole object or a fixed collection of rigid parts when manipulated by a robot; modeling objects with topology changes, *i.e*., object fragmentation in a cutting task, is still largely unexplored.

The challenges of modeling object fragmentation are twofold: (i) An object or its fragments exhibit considerable variation in terms of their *configurations* (*i.e*., the layout, fragment number, pose, and shape of each fragment) during fragmentation. (ii) An object fragmentation process intrinsically involves one-to-many transitions (*i.e*., an object breaking into multiple fragments), and there are many ways an object might be potentially fragmented. As a result, it is nontrivial to find a proper state representation, hindering the direct employment of methods such as neural networks [HLF19, VCC20], probabilistic graphical models [LZZ19, EGL19], symbolic logic [KL11, JZW21b], *etc*., to model such complex causal transitions during fragmentation. To overcome these challenges, a desired state representation should be reconfigurable and extendable to account for the drastic variations in object fragmentation while being abstract enough to reduce the number of possible transitions for efficient planning.

In this work, based on the stochastic grammar model, we develop a *fluent* notation [New36] to represent the state of an object–or that of its fragments–during cutting and derive a *fluent space* to describe the possible fragmentation (*i.e*., the states and their causal transitions). Being successful in modeling scenes [ZM07, HQZ18, QZH18] and dynamic events [EGL19, QJH20, ZZZ15], a stochastic grammar consists of a set of production rules that generate terminal or non-terminal variables from existing non-terminal ones, akin to the process of an object breaking into pieces—the original object generates newly appeared fragments. Specifically, the *grammar* itself incorporates all possible states an object may finally be as fragmentation repeats, and the *production rules* of the grammar indicate all valid one-to-many transitions. Together, they form the fluent space of object cutting. Furthermore, each *parse tree* derived from the grammar reflects a specific fragmentation process produced by a sequence of cutting actions, whose *terminal nodes* correspond to fragments in the resulting configuration and collectively define the resulting fluent.

Fig. 6.1 presents the proposed stochastic grammar that models the object fragmentation process

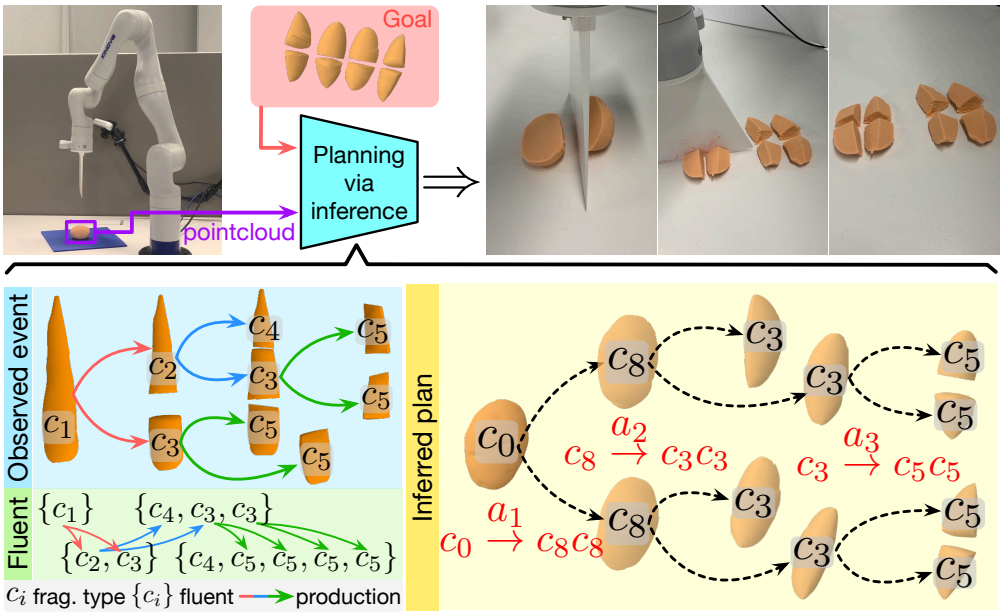


Figure 6.1: **Planning for object cutting with a stochastic grammar of object fragmentation.** The grammar reveals the underlying fluent space of object fragmentation and captures causal transitions in a compositional manner with production rules. An observed fragmentation process is represented as a parse tree derived from the grammar; planning for object cutting is to infer an optimal parse tree that describes the desired fragmentation. Observing cutting a carrot could support planning actions for cutting a potato into the same by sharing the production rule $c_3 \rightarrow c_5 c_5$.

of cutting a carrot into chunks and supports planning cutting actions for a novel situation of cutting a potato into the same. We extract shape features for fragments and cluster them to obtain a much smaller set of variables to represent fluents and to induce production rules that describe the causal transitions between fluents. Crucially, the cluster number is determined such that the resulting grammar seeks to reduce its complexity by having fewer types of variables while preserving the necessary discriminability of fragments for consistency of transitions. More importantly, grammar's recursive and compositional nature allows us to model the fluent and fluent space compactly and flexibly and achieve better generalization.

In the experiments, we collect a dataset of human cutting demonstrations in simulation, from which we induce the grammar model and learn to generate action parameters for cutting. Dur-

ing the test phase, we demonstrate the efficacy of the proposed grammar-based representation and planning method on a series of object-cutting tasks, including those under novel setups. A preliminary real robot experiment also shows that our method can be applied to real-world object-cutting scenarios.

6.1.1 Related Work

Planning a sequence of actions to alter objects' states towards a goal is a long-standing problem in robotics and artificial intelligence. Task planning [LaV06] efficiently searches for a sequence of discrete actions to reach the goal based on a known planning domain, usually defined in PDDL [AHK98]. While this approach provides a general and practical solution, it is limited to a fixed number of objects and relies on known transition models and hand-crafted state and action abstractions [SFR14, Tou15, GLK18, JZW21b].

An alternative approach is Model-based Reinforcement Learning, which effectively learns a transition model from interaction data, potentially with a learned state representation [CCM18, JFZ19, HLF19]. While this method achieves impressive performance, it requires extensive exploration and may not generalize well in complex scenarios.

We adopt an approach inspired by Task and Motion Planning (TAMP), accommodating a changing number of objects, while learning a stochastic grammar model to represent an abstract planning domain for object cutting. Notably, the production rules in the grammar model effectively bridge the gap between planning discrete cutting actions and generating continuous action parameters (see Section 6.3.3).

6.1.2 Overview

The remainder of this chapter is organized as follows. Section 6.2 formally models the object fragmentation process in object cutting using stochastic grammar and provides insights into learning such a model from human demonstrations. Section 6.3 formulates the plan-to-cut problem

as probabilistic inference, leveraging the learned grammar model, and presents an algorithm for online planning. Furthermore, in Section 6.4, we demonstrate the efficacy of our method through a series of experiments, including novel scenarios. Finally, we conclude the chapter and discuss future research directions in Section 6.5.

6.2 Attributed Stochastic Grammar of Object Fragmentation

An object fragmentation process $r_o : \Omega_o \rightarrow \Omega_o$ transforms a set of object fragments $\mathcal{I}^{\text{pre}} \in \Omega_o$ into another set of fragments $\mathcal{I}^{\text{post}} \in \Omega_o$, where $\mathcal{I} = \{o_i\}$ represents the configuration of the object fragments, $1 \leqslant |\mathcal{I}^{\text{pre}}| \leqslant |\mathcal{I}^{\text{post}}|$, and o_i represents an initial whole object or a fragment by its shape (*e.g.*, point cloud), pose, *etc*. Considering the complex nature of Ω_o, where each fragment could vary in shape, we instead regard some fragments $o_i \in \mathcal{I}$ as the same type $c_j \in C$ via clustering, where $S = \{c_j\}$ defines an object fluent of the configuration $\mathcal{I}$. As such, we obtain a simplified fluent space $\Omega_s = \{S\}$ that depicts a fragmentation process $r_s : \Omega_s \rightarrow \Omega_s$ with better abstraction.

6.2.1 Grammar representation of object fragmentation

We adopt an attributed stochastic grammar [PNZ17] to model causal transitions in object fragmentation, where terminal variables with their attributes represent the configuration of fragments, and production rules capture the valid causal transitions that an object breaks into multiple fragments. Formally, the attributed stochastic grammar is defined by a 5-tuple $\mathcal{G} = \langle V_{NT}, V_T, v_S, R, \mathbb{P} \rangle$, where $v_{NT} \in V_{NT}$ is a non-terminal variable that denotes a fragment type $c \in C$, $v_T \in V_T$ is a terminal variable that denotes a fragment type $c \in C$ with pose $q \in SE(3)$ and shape feature z as its attributes, v_S is the start symbol, $\mathbb{P}$ is the probability of the production rules defined over the grammar, and $r_i \in R$ is the production rule $r_i : V_{NT} \rightarrow (V_{NT} \cup V_T)^*$, where $(\cdot)^*$ is the Kleene star operation, enabling a production rule to describe an arbitrary fragmentation within the domain of $V_{NT} \cup V_T$. A fluent S is defined by terminals of a parse tree pt generated from $\mathcal{G}$, and the fluent space is defined by $\Omega_s = L(\mathcal{G})$, where $L(\mathcal{G})$ represents the set of all possible fluents generated by

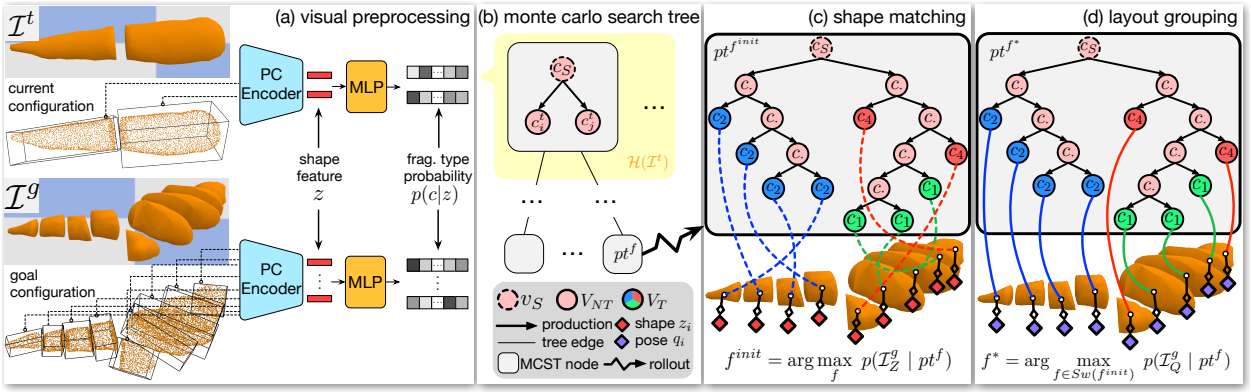


Figure 6.2: **An illustration of the inference process to obtain an optimal parse tree pt^* through MCTS.** (a) Given fragment point clouds in the current or goal configuration, we extract a shape feature for each fragment with a pre-trained point cloud encoder and process it with an MLP to classify the fragment type $p(c|z)$ (the vector shows probability in greyscale). (b) We show an example of a Monte Carlo search tree where the state of a search node is a parse tree derived from the grammar. The expansion of a search node is to apply production rules to its parse tree. The yellow region $\mathcal{H}(\mathcal{I}^t)$ is a set of search nodes whose states (*i.e.*, parse trees) are sampled from fragments in $\mathcal{I}^t$ according to $p(c|z)$. (c-d) To evaluate rollout results, we find the best assignment that grounds each terminal node to a fragment in $\mathcal{I}^g$. The dotted lines in (c) represent an optimal assignment that maximizes the shape matching likelihood in Eq. (6.5), which is further refined to maximize the layout grouping likelihood in Eq. (6.6), shown in solid lines in (d).

$\mathcal{G}$. Intuitively, a parse tree pt derived from $\mathcal{G}$ represents a plausible fragmentation sequence: the collection of terminals corresponds to the resulting fragments, and the non-terminals indicate the intermediate fragments in the past that subsequently fragment into the final configuration due to the sequence of applied productions (*i.e.*, cutting actions).

6.2.2 Grammar induction from human demonstrations

We propose to learn the stochastic grammar from object-cutting sequences generated by human demonstrations; please refer to Section 6.4.1 for details of data collection.

Corpus generation We extract a shape feature z for an object or fragment $o_i \in \mathcal{I}$ using a pre-trained point cloud encoder (see Section 6.3.4), and cluster all features $\{z\}$ into k fragment types $\{c\}$. Then a corpus $\mathcal{D}_c^k = \left\{c_i^{\text{pre}} \rightarrow \{c_{i,j}^{\text{post}}\}\right\}$ is obtained by recording the fragment type before and after each cutting action.

A critical question is how to determine the proper number of fragment types k to reduce grammar complexity while maintaining sufficient discriminability among fragments. We solve it by balancing the data likelihood and model complexity in grammar induction; see the details below.

Grammar induction Given corpus $\mathcal{D}_c^k$, we use MAP estimation to induce an optimal grammar:

$$\begin{aligned} \mathcal{G}^* &= \underset{\mathcal{G}^k}{\operatorname{argmax}}\ p(\mathcal{D}_c^k \mid \mathcal{G}^k)\, p(\mathcal{G}^k) \\ &= \underset{\mathcal{G}^k}{\operatorname{argmax}} \underbrace{\prod_{(\alpha_i \rightarrow \beta_i) \in \mathcal{D}_c^k} p(\alpha_i \rightarrow \beta_i \mid \mathcal{G}^k)}_{\text{data likelihood}} \cdot \underbrace{e^{\gamma |\mathcal{G}^k|}}_{\text{model prior}}, \end{aligned} \tag{6.1}$$

where $\alpha_i \rightarrow \beta_i$ is the i-th production in $\mathcal{D}_c^k$, γ a scalar coefficient, $|\mathcal{G}^k|$ the model size only depending on k, and $p(\alpha_i \rightarrow \beta_i \mid \mathcal{G})$ the branching probability of the production $\alpha_i \rightarrow \beta_i$ defined in $\mathbb{P}$.

We adopt an iterative non-parametric clustering approach, similar to DP-means [KJ12], to solve for $\mathcal{G}^*$ in Eq. (6.1) by alternating two steps: search for a better k, and estimate the best production rules. With a fixed k, the best production rule probability aligns with the frequency of each alternative choice [ZM07]:

$$p(\alpha \rightarrow \beta_i) = \#(\alpha \rightarrow \beta_i) / \sum_{j=1}^{n(\alpha)} \#(\alpha \rightarrow \beta_j), \tag{6.2}$$

where $\#(\alpha \rightarrow \beta)$ is the number of productions following $\alpha \rightarrow \beta$ in the corpus, and $n(\alpha)$ is the number of productions whose left-side (the non-terminals) is α. For ease of planning with $\mathcal{G}^*$, we also fit a classifier on the clustered fragments to model $p(c|z)$, the probability of a fragment's type c given its shape feature z.

6.3 Planning for Object Cutting

We aim to plan a sequence of cutting actions to achieve the goal configuration $\mathcal{I}^g$ from an initial configuration of fragments $\mathcal{I}^t$. Each cutting action involves cutting one object or fragment using a 3D cutting plane represented as $\boldsymbol{\pi} = [\boldsymbol{n}^T, d]^T \in \mathbb{R}^4$, where $\|\boldsymbol{n}\|_2 = 1$ is a unit plane normal vector, and $\boldsymbol{n}^T \cdot \boldsymbol{v} + d = 0$ represents the cutting plane constraint. We represent cutting planes in the canonical frame of the fragment to cut.

The planning problem is transformed into inferring an optimal parse tree of desired fragments given the learned grammar model that captures all possible causal transitions in object cutting. The planning is solved online using MCTS, detailed in Section 6.3.2. Each production in the parse tree corresponds to a cutting action, and given the inferred parse tree, we generate the cutting plane $\boldsymbol{\pi}$ for each action with a sampling-based method (see Section 6.3.3).

6.3.1 The posterior probability of parse trees

We derive the posterior probability of a parse tree pt, representing a fragmentation sequence or a plan of cutting actions, given the goal configuration $\mathcal{I}^g$ and the grammar $\mathcal{G}$. For each fragment in $\mathcal{I}^g$, we extract shape feature z and pose q, resulting in $\mathcal{I}^g_Z = \{z_i\}$ and $\mathcal{I}^g_Q = \{q_i\}$.

The posterior probability is given by:

$$p(pt \mid \mathcal{I}^g, \mathcal{G}) \propto \underbrace{p(pt \mid \mathcal{G})}_{\substack{\text{grammar} \\ \text{prior}}} \underbrace{p(\mathcal{I}^g_Z \mid pt)}_{\substack{\text{shape matching} \\ \text{likelihood}}} \underbrace{p(\mathcal{I}^g_Q \mid pt)}_{\substack{\text{layout grouping} \\ \text{likelihood}}}, \tag{6.3}$$

where the first term is the prior probability of the parse tree pt given $\mathcal{G}$, and the second and third terms describe the likelihood of observing $\mathcal{I}^g$ given pt in terms of fragment shape and pose. The overall posterior probability measures the alignment between pt generated by $\mathcal{G}$ and the goal configuration $\mathcal{I}^g$.

Grammar prior The grammar prior captures possible causal transitions of object or fragment types. It is based on the learned production rules and branching probability:

$$p(pt \mid \mathcal{G}) = \prod_{(\alpha_i \to \beta_i) \in R^{pt}} p(\alpha_i \to \beta_i \mid \mathcal{G}), \tag{6.4}$$

where R^{pt} is the set of productions in the parse tree pt, and $p(\alpha_i \to \beta_i \mid \mathcal{G})$ is the conditional probability of choosing the production $\alpha_i \to \beta_i$ given the non-terminal node α_i.

Shape matching likelihood The shape matching term evaluates the alignment between pt and the goal configuration $\mathcal{I}^g$ in terms of fragment geometry:

$$p(\mathcal{I}_Z^g \mid pt) = \prod_{i=1}^{N} p(z_i \mid c_i) \propto \prod_{i=1}^{N} p(c_i \mid z_i)\, p(z_i), \tag{6.5}$$

where c_i is the fragment type of the i-th terminal node in pt, z_i is the shape feature of the corresponding fragment, and N is the number of fragments in $\mathcal{I}_Z^g$. The prior $p(z_i)$ is a normal distribution fitted on the train set, and $p(c_i|z_i)$ is obtained from the classifier based on the shape feature z_i.

Layout grouping likelihood The layout grouping term measures the alignment between pt and $\mathcal{I}^g$ in terms of fragment layout:

$$\begin{aligned} p(\mathcal{I}_Q^g \mid pt) &= \prod_{(\alpha_i \to \beta_i) \in R^{pt}} p\left(\beta_i \mid \alpha_i \to \beta_i\right) \\ &= \prod_{(\alpha_i \to \beta_i) \in R^{pt}} \prod_{v_j^{\beta_i} \in \beta_i} p\left(v_j^{\beta_i} \mid \alpha_i \to \beta_i\right), \end{aligned} \tag{6.6}$$

where $\alpha_i \to \beta_i$ is the i-th production in R^{pt}, α_i is the non-terminal node being expanded, and β_i represents the produced nodes from the rule. $v_j^{\beta_i}$ is the j-th produced node in β_i, and $p(v_j^{\beta_i}|\alpha_i \to \beta_i)$ gives the probability that production $\alpha_i \to \beta_i$ produces node $v_j^{\beta_i}$.

Assuming that the closer the fragments, the more likely they come from the same piece, we define the distribution $p(v_j^{\beta_i} \mid \alpha_i \to \beta_i)$ by an energy function:

$$p\left(v_j^{\beta_i} \mid \alpha_i \to \beta_i\right) = \frac{1}{Z} \exp\left(-\text{dist}(q^{\alpha_i}, q_j^{\beta_i})\right), \tag{6.7}$$

where Z is the partition function, $q_j^{\beta_i}$ the averaged pose of fragments in descendants under the node $v_j^{\beta_i}$, q^{α_i} the averaged poses of descendants in α_i, and $\text{dist}(\cdot,\cdot)$ the distance function that measures the distance between two poses. In practice, we calculate the Euclidean distance between the positions of two nodes and adopt dynamic programming when computing q^{α_i} and $q_j^{\beta_i}$ to avoid redundant computations.

6.3.2 Inference of the optimal parse tree

Given the current configuration $\mathcal{I}^t$, we aim to plan an optimal sequence of cutting actions that leads to a desired configuration $\mathcal{I}^g$. We formulate the planning process as inferring the optimal parse tree via an MAP estimate:

$$\begin{aligned} pt^* &= \underset{pt \in \mathcal{H}(\mathcal{I}^t)}{\text{argmax}}\ p(pt \mid \mathcal{I}^g, \mathcal{G}) \\ &= \underset{pt \in \mathcal{H}(\mathcal{I}^t)}{\text{argmax}}\ p(pt \mid \mathcal{G})\, p(\mathcal{I}_Z^g \mid pt, \mathcal{G})\, p(\mathcal{I}_Q^g \mid pt, \mathcal{G}), \end{aligned} \tag{6.8}$$

where $\mathcal{H}(\mathcal{I}^t)$ is a set of parse trees whose expansions from the start variable are sampled from $p(c|z)$ for each fragment in $\mathcal{I}^t$ after extracting shape feature z.

Since the computation of pt^* in Eq. (6.8) is intractable, we approximate pt^* via Monte Carlo Tree Search (MCTS) as shown in Fig. 6.2b. Initially, the algorithm starts with the root node of the search tree, which contains the start variable v_S of the grammar. The expansion and simulation step of MCTS is a process of applying feasible production rules (*i.e.*, possible causal transitions) on the parse tree of the search node, and the rollout results in each round are evaluated by measuring the objective function in Eq. (6.8). During the backpropagation step, we use the objective function value as the score to update the nodes on the path from the root to the rollout result. Finally, the best rollout result among all rounds in MCTS will be selected as pt^*.

To evaluate the objective function, we need to align every terminal node with a unique fragment in $\mathcal{I}^g$, as described in Section 6.3.1. Hence, for the i-th round of rollout, we compute an optimal assignment function $f_i^* : V_T \to O$ that grounds each terminal node v_T in pt_i to a unique fragment o in $\mathcal{I}^g$, such that the resulting parse tree $pt_i^{f^*}$ maximizes the objective in Eq. (6.8) as well. Since

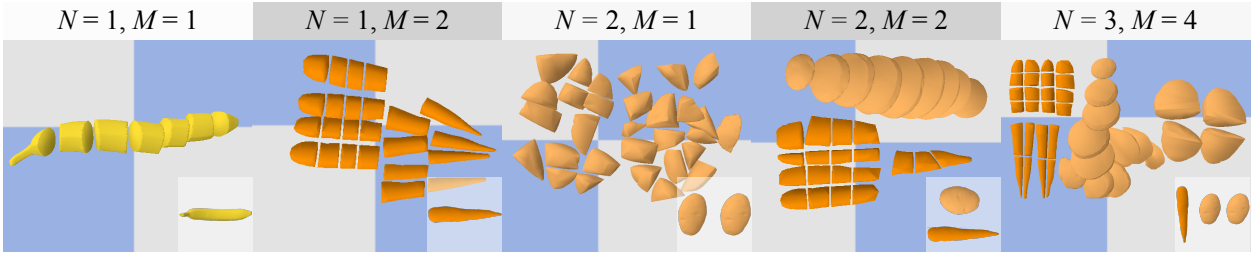


Figure 6.3: **Examples of collected data with different levels of task complexity.** N is the initial number of objects, and M the number of fragment categories in the goal configurations. The bottom right corner of each subfigure shows the initial configuration.

the grammar prior term is irrelevant to the assignment, we have:

$$f^* = \underset{f}{\operatorname{argmax}}\ p(\mathcal{I}_Q^g \mid pt_i^f)\, p(\mathcal{I}_Z^g \mid pt_i^f), \tag{6.9}$$

where pt_i^f denotes the parse tree whose terminal nodes are grounded to fragments in $\mathcal{I}^g$ by the assignment function f.

Since directly computing f^* is intractable (factorial to the number of fragments), we obtain an approximate solution in two steps: (i) Compute an assignment function f^{init} that maximizes the shape matching likelihood $p(\mathcal{I}_Z^g \mid pt^f)$ in Eq. (6.5); see the dotted lines in Fig. 6.2c. (ii) Refine f^{init} into f^* that maximizes the layout grouping likelihood $p(\mathcal{I}_Q^g \mid pt^f)$ in Eq. (6.6) while conserving the optimality obtained in the previous step; see solid lines in Fig. 6.2d.

The first step formulates a linear assignment problem that can be solved in polynomial time using the Hungarian algorithm [Kuh55]. Then we adopt the simulated annealing algorithm [KGV83] to maximize $p(\mathcal{I}_Q^g \mid pt^f)$, where we randomly swap the matched terminal nodes of two fragments. As we do not want to violate the established optimality of $p(\mathcal{I}_Z^g \mid pt^f)$, we only swap terminal nodes with the same fragment type.

6.3.3 Cutting plane generation

With an inferred optimal parse tree whose productions correspond to cutting actions, we generate cutting planes to make the actions executable. We model the cutting plane $\boldsymbol{\pi}$ of an action as a Gaussian Mixture Model (Gaussian Mixture Model (GMM)) with its parameters depending on the production rule r and the shape feature z of the fragment to cut. The GMM parameters are regressed using a two-layer Mixture Density Network [Bis94], learned from human demonstration data. During planning, we compute GMM parameters with a forward pass of the neural network and sample cutting planes from the corresponding GMMs for execution. We generate cutting planes for each production separately (*i.e.*, cut one object or fragment at a time).

6.3.4 Implementation details

We define a consistent *canonical frame* for each fragment to match and distinguish between object fragments presented in different poses. The canonical frame is defined on a shape so that its projection along the z-axis is maximized, its projection along the x-axis is minimized, and its volume in the first octant is the largest. In practice, we compute the canonical frame of a fragment by principal component analysis.

We use a 17-dimensional vector $z = (z_{\text{shape}}, z_{\text{scale}})$ as the *shape feature*, where $z_{\text{shape}} \in \mathbb{R}^{16}$ encodes the normalized shape represented in its canonical frame with a point cloud encoder. The scalar $z_{\text{scale}} \in \mathbb{R}$ represents the scale. We adopt a naive encoder that processes all point coordinates and normals with a shared Multi-layer Perceptron (MLP) followed by an average pooling layer; the encoder is trained on all fragments in the train set of human cutting data following IMNet [CGG19].

6.4 Simulations and Experiments

We developed an object-cutting simulator based on BulletPhysics [CB21] to collect human demonstrations and test our method and baselines. Specifically, we implemented a Slice function in BulletPhysics that slices an object with a 3D plane. All methods were trained and evaluated in the simulator. Furthermore, we demonstrated that our proposed model, trained in the simulation environment, can effectively handle real-world object-cutting tasks a physical robot executes.

6.4.1 Data preparation

To collect human demonstrations, we asked human subjects to cut virtual objects presented in the simulator into one of the four fragment categories (*i.e.*, chunks, slices, cubes, and strips) or their combinations, using an intuitive Graphical User Interface (GUI) offered by the simulator. A cutting action is applied as a human subject specifies a 3D cutting plane by clicking two points on the GUI. We recorded each trail of demonstration as a sequence of fragment configurations and cutting actions; the ground-truth 3D geometry of each fragment and its pose can be directly retrieved from the simulator. A total of 110 object-cutting trails were collected and partitioned according to the initial number of objects N and the number of fragment categories in the goal configurations M; see Fig. 6.3 for some examples. We split the collected data, using a subset (40%) of $N = 1, M = 1$ as the train set and test on the remaining trails (*i.e.*, the rest of partition $N = 1, M = 1$ and partitions $N > 1, M > 1$).

6.4.2 Experimental setup

We test our method against various baselines to compute a sequence of cutting actions that reach a goal configuration $\mathcal{I}_g$ from a current fragment configuration $\mathcal{I}_t$, retrieved from test set trials. Instead of planning and executing all actions at once, we execute one action at a time and re-plan from the resultant configuration. This process repeats until we achieve the target number of fragments in the goal configuration. In our approach, we randomly select one production and

Table 6.1: **Quantitative results of planning for object cutting under various task setups.** We evaluate all methods using the best-matched IoU and Human Rating (HR) on test sets with different N, M combinations, averaged across five runs; $\pm$ denotes standard deviation.

Task Setup		BC		QNet		Ours		Human	
		IoU	HR	IoU	HR	IoU	HR	IoU	HR
Seen	N=1, M=1	0.37 ± 0.11	2.19 ± 1.07	0.40 ± 0.16	2.14 ± 1.21	$\mathbf{0.58} \pm 0.08$	$\mathbf{4.32} \pm 0.77$	0.57 ± 0.03	4.48 ± 0.96
Unseen	N=1, M=2	0.35 ± 0.08	1.76 ± 0.87	0.32 ± 0.12	1.95 ± 0.87	$\mathbf{0.49} \pm 0.06$	$\mathbf{3.60} \pm 1.02$	0.62 ± 0.07	4.86 ± 0.35
	N=2, M=1	0.44 ± 0.08	1.64 ± 0.65	0.34 ± 0.16	1.19 ± 0.39	$\mathbf{0.56} \pm 0.03$	$\mathbf{3.69} \pm 0.89$	0.62 ± 0.09	4.83 ± 0.37
	N=2, M=2	0.42 ± 0.03	2.07 ± 0.86	0.29 ± 0.09	1.24 ± 0.43	$\mathbf{0.52} \pm 0.04$	$\mathbf{3.74} \pm 0.90$	0.56 ± 0.04	4.79 ± 0.56
	N=2, M=3	0.38 ± 0.03	1.73 ± 0.99	0.28 ± 0.09	1.52 ± 0.92	$\mathbf{0.52} \pm 0.03$	$\mathbf{3.21} \pm 0.86$	0.60 ± 0.04	4.81 ± 0.55
	N=3, M=4	0.38 ± 0.04	1.57 ± 0.62	0.22 ± 0.08	1.26 ± 0.49	$\mathbf{0.52} \pm 0.02$	$\mathbf{3.21} \pm 0.86$	0.56 ± 0.04	4.81 ± 0.55

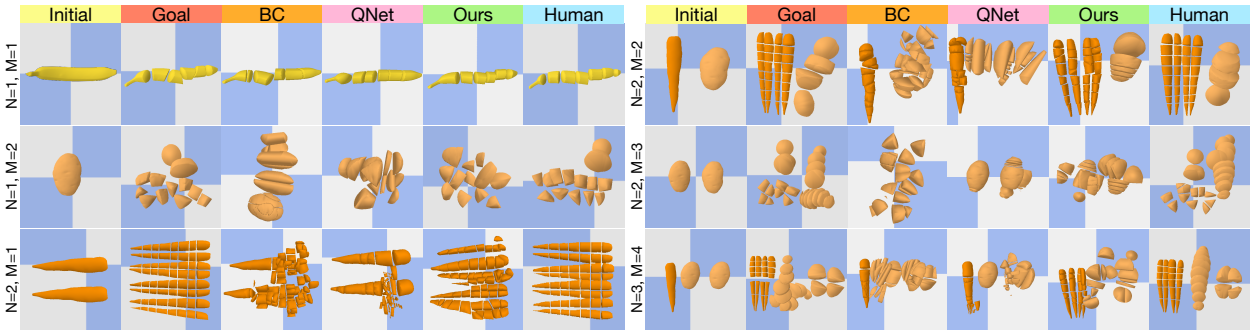


Figure 6.4: **Qualitative results of object cutting to achieve a desired set of fragments.** Each row shows sample results of different methods under specific task setups.

execute the corresponding cutting action when multiple productions are available from the derived parse tree. Below, we describe the baseline methods and evaluation metrics that measure goal achievement.

6.4.2.1 Baselines

Given the success of learning-to-plan methods in handling complex state spaces, we design two baselines using a *state embedding* s, obtained by projecting shape features with an MLP followed

by sum-pooling: (i) **Behavioral Cloning (BC)**: It learns a goal-directed policy with a two-layer MLP to mimic human actions in collected demonstrations. The policy predicts the cutting probability per fragment based on its shape feature z_i, current state embedding s, and goal state embedding s_g. The fragment to be cut is sampled based on the obtained probability, and cutting planes are sampled from a learned GMM conditioned on z_i and s_g. (ii) **Offline Deep Q Network (QNet)**: This model-free reinforcement learning approach is trained on logged data. We approximate a goal-conditioned action value function (Q function) using a two-layer MLP. During training, we assign sparse rewards to state-action pairs that achieve the goal, where an action is to choose a fragment to cut. Since the train set only contains positive demonstrations, we add a large margin loss term to encourage assigning a higher value to actions seen during training [KS20]. At test time, we select the best fragment to cut according to the Q function and sample a cutting plane identical to BC. Furthermore, we recruit (iii) **Human** participants to perform cutting tasks under the same setup, which serves as the performance upper bound.

6.4.2.2 Evaluation metrics

Due to geometric similarities between two fragment configurations despite different layouts, we design two metrics to evaluate how well the produced fragments match the goal configuration: (i) **Mean best-matched IoU**: This *objective* metric is the averaged IoU between the best-matched fragment pairs (see Fig. 6.5 dotted lines) in the produced final fragments and fragments in the goal configuration. We compute best-matched fragment pairs as a linear assignment problem using the Hungarian algorithm [Kuh55] in polynomial time. (ii) **Human Rating**: We recruit human participants to *subjectively* rate the fitness of the produced fragments against the goal. The rating ranges from 1 to 5 in discrete values, with higher scores indicating a better match.

6.4.3 Simulated results

We present the results of our method and the baselines under different test setups in Table 6.1, and a qualitative comparison in Fig. 6.4. Our method outperforms BC and QNet in both objective and subjective metrics across all six setups, demonstrating superior generalization capabilities in scenarios involving more objects to cut ($N > 1$) and/or a composition of fragment types in the goal ($M > 1$). While BC performs on par with QNet in relatively simple task instances ($N = 1, M = 1$), it outperforms QNet in most generalization setups. Our method excels in learning a valid planning domain with a small amount of data and generalizing to novel task setups, thanks to the grammar model that effectively abstracts the fluent space and represents causal transitions in a compositional manner. The qualitative results and superior human rating of our method further demonstrate that the grammar models the fluents and causal transitions in a semantically meaningful way, aligning well with humans' mental abstraction of fragmenting objects.

6.4.4 Real-world robot experiment

We conduct a real robot experiment with a Kinova Gen 3 manipulator with a knife-like end-effector to cut an object into desired fragments. The goal configuration is given as fragment point clouds,

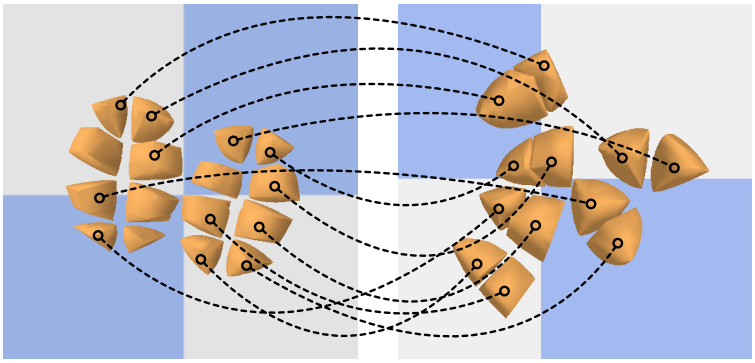


Figure 6.5: **An example of best-matched fragments for evaluation.** Each dotted line connects a pair of best-matched fragments. Since the number of fragments is unbalanced between two sets of fragments, some fragments in the larger set remain unmatched.

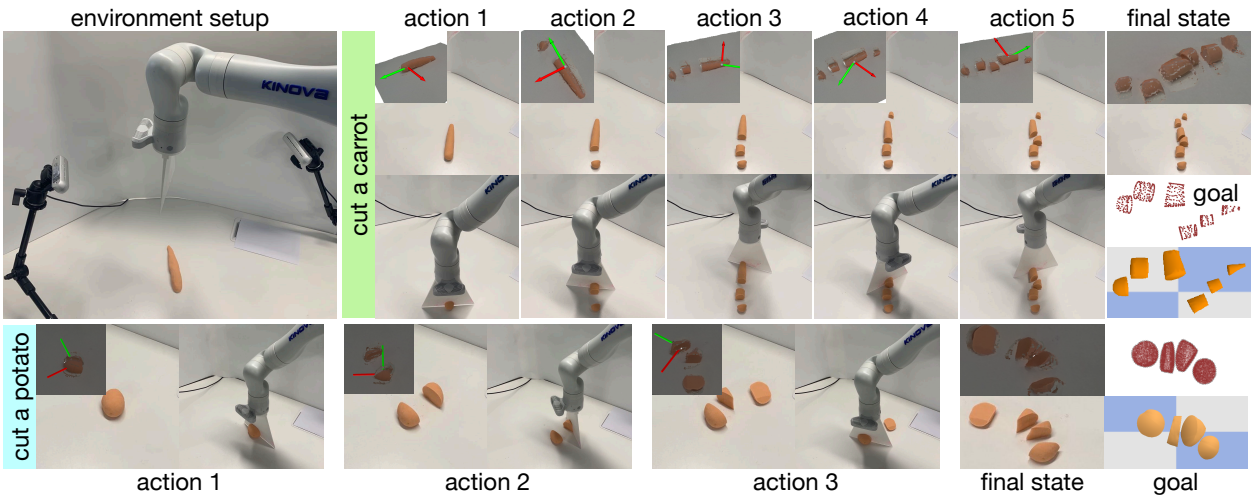


Figure 6.6: **Real-world object cutting experiment.** Experiment on object cutting with a Kinova Gen 3 manipulator. The top-left figure illustrates the environment setup, and examples of the robot cutting a carrot and a potato are shown. The sequence of actions applied and the resulting fragments demonstrate a good alignment with the goal.

while the robot observes a single point cloud of the current configuration by fusing outputs from two third-person-view depth cameras. The observed point cloud is segmented to produce per-fragment point clouds using a point cloud segmentation model [QYS17] trained on the simulation dataset. We plan and execute a sequence of cutting actions following Section 6.3. The grammar and neural networks are fully trained on simulated data as described in Section 6.4.3.

Fig. 6.6 presents the environment setup and keyframes of the robot executing planned actions to cut a carrot and a potato. The robot can generate meaningful cutting actions and produce fragments well-aligned with the goal. The experimental results demonstrate our method's ability to handle perception uncertainties and its potential in real-world object-cutting tasks.

6.5 Conclusion

In this work, we introduced a stochastic grammar of object fragmentation, which abstracts the state of fragments as terminal variables and accommodates causal transitions in object fragmentation through production rules. The proposed representation is powerful for modeling causal transitions in object fragmentation, enabling an agent to plan actions that cut an object into desired fragments. The planning problem is formulated as inferring an optimal parse tree of the desired configuration, where terminal nodes are ground to the produced final fragments and the productions indicate cutting actions. Our method achieved remarkable performance in planning for object-cutting tasks, even when applied to novel test setups with significant variations compared to the training set. Moreover, we conducted a preliminary real robot experiment utilizing a model trained in simulation, demonstrating the robustness of our method in the physical world. This work introduces a new perspective on object modeling and explores a new dimension of robot manipulation capability.

Limitations While our method effectively abstracts the state space with fluents and models a simplified set of causal transitions, it still has some limitations. One notable limitation is the computational complexity of the MCTS method, especially when the goal configuration involves a large number of fragments, MCTS often requires considerable time to arrive at an optimal solution. Integrating Reinforcement Learning techniques with tree search, similar to Alpha-Go [SHM16], could be a potential avenue to address this issue and further improve planning efficiency. Additionally, our method only permits cutting a single object in its object-centric frame. Extending the proposed approach to cut multiple objects simultaneously and incorporating interactions between objects during cutting remain an open challenge for future research.

Discussion One of the key strengths of our approach is the ability to generalize to unseen scenarios and handle various cutting tasks with different numbers of objects and fragment categories. The grammar model, representing fragments as terminal variables and causal transitions through productions, allows the agent to abstract the object's state and plan for actions accordingly. This

enables our method to effectively infer an optimal parse tree that guides the cutting process toward the goal configuration. Moreover, our approach is capable of learning the planning domain from a relatively small amount of simulated data, which makes it a practical and efficient solution for real-world object-cutting tasks on a physical robot.

Furthermore, the real-world experiment conducted with a Kinova Gen 3 manipulator showcased the applicability of our method in a physical setting. The robot was able to generate meaningful cutting actions and produce fragments that aligned well with the desired configuration. However, limitations in the modeling of complex contact dynamics could introduce uncertainty in the execution phase. Addressing these challenges and achieving more precise execution in real-world scenarios will be critical for practical deployment.

CHAPTER 7

Conclusion

This dissertation introduces a unified scene abstraction perspective for long-horizon robot planning with generalization to diverse tasks and scenarios. The core of this approach is to acquire a perception model that abstracts raw observations into structural representations, and a world model that allows predicting action outcomes for search-based planning. We focus on scene graph-based representations that abstract functional objects and their relations as symbols, which span a compact state space for effective planning. We present three parts of our work that tackle challenges of different aspects.

- Chapter 2 focuses on the **perception** problem of reconstructing a 3D scene graph representation for robot TAMP. We show that the proposed perception system can robustly recover 3D objects and their relations from observation sequences, establishing foundations for robot planning with the scene graph representation.

- Chapter 3 and Chapter 4 present closed-loop **planning** solutions that allow robots to reason about and recover from failures. We show that by leveraging internal knowledge of language models and environment feedback such as scene graphs, we achieve open-ended reasoning and planning in a large range of mobile and tabletop manipulation tasks.

- Chapter 5 and Chapter 6 introduce methods for **learning** symbolic abstractions and world models for effective and generalizable robot planning. We show that by learning relational symbols (*i.e.*, predicates) and object symbols and the corresponding world models, our framework yields strong generalizability in unseen object configurations and goals.

Below, we summarize several key insights and promising future directions in building robots capable of solving unseen long-horizon tasks from raw sensor observations.

Perceptual Abstraction for Task Generalization Building perceptual abstractions that generalize is the backbone of task generalization. This dissertation develops perception solutions on top of general-purpose perception models (*e.g.*, object detection, segmentation), which possess a certain level of generalization ability. Moving forward, a promising direction is to develop pretraining methods that learn generalizable perceptual abstractions, *i.e.*, most likely dense embeddings, for visual and other sensor modalities in robotic tasks.

Robustness and Generlizability of Manipulation Skills An important assumption of the proposed scene abstraction framework and many robot planning frameworks is access to robust manipulation skills. However, this assumption usually does not hold for tasks beyond picking up and placing simple objects. Learning manipulation skills as neural policies from data is a promising direction to explore. While scaling up data generation in the real world is expensive, one possibility is to generate large-scale simulation data with drastic domain randomization.

REFERENCES

[ABB22] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. "Do as i can, not as i say: Grounding language in robotic affordances." *arXiv:2204.01691*, 2022.

[ADD19] Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X Chang, and Matthias Nießner. "Scan2cad: Learning cad model alignment in rgb-d scans." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[ADN19] Armen Avetisyan, Angela Dai, and Matthias Nießner. "End-to-end cad model retrieval and 9dof alignment in 3d scans." In *International Conference on Computer Vision (ICCV)*, 2019.

[AHG19] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. "3d scene graph: A structure for unified semantics, 3d space, and camera." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[AHK98] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. "PDDL— The Planning Domain Definition Language." Technical report, Yale Center for Computational Vision and Control, 1998.

[Asa19] Masataro Asai. "Unsupervised grounding of plannable first-order logic representation from images." In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 583–591, 2019.

[ASP22] Alper Ahmetoglu, M Yunus Seker, Justus Piater, Erhan Oztop, and Emre Ugur. "Deepsym: Deep symbol generation and rule learning for planning from unsupervised robot interaction." *Journal of Artificial Intelligence Research*, **75**:709–745, 2022.

[AXW21] William Agnew, Christopher Xie, Aaron Walsman, Octavian Murad, Yubo Wang, Pedro Domingos, and Siddhartha Srinivasa. "Amodal 3d reconstruction for robotic manipulation via stability and connectivity." In *Conference on Robot Learning (CoRL)*, 2021.

[Bis94] Christopher M Bishop. "Mixture density networks." Technical report, Aston University, 1994.

[BMR20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners." *Advances in Neural Information Processing Systems (NeurIPS)*, **33**:1877–1901, 2020.

[BPM11] Léo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse, and Eiichi Yoshida. "Real-time replanning using 3D environment for humanoid robot." In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 584–589. IEEE, 2011.

[BPY22] Andreea Bobu, Chris Paxton, Wei Yang, Balakumar Sundaralingam, Yu-Wei Chao, Maya Cakmak, and Dieter Fox. "Learning perceptual concepts by bootstrapping from human queries." *IEEE Robotics and Automation Letters (RA-L)*, **7**(4):11260–11267, 2022.

[CAZ23] Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. "Autotamp: Autoregressive task and motion planning with llms as translators and checkers." *arXiv:2306.06531*, 2023.

[CB21] Erwin Coumans and Yunfei Bai. "PyBullet, a Python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2016–2021.

[CCC16] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age." *IEEE Transactions on Robotics (T-RO)*, **32**(6):1309–1332, 2016.

[CCM18] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[CD17] Hyung Jin Chang and Yiannis Demiris. "Highly articulated kinematic structure estimation combining motion and skeleton information." *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, **40**(9):2165–2179, 2017.

[CFG15] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository." *arXiv preprint arXiv:1512.03012*, 2015.

[CFK22] 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." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1940–1946. IEEE, 2022.

[CGG19] Tao Chen, Saurabh Gupta, and Abhinav Gupta. "Learning exploration policies for navigation." In *International Conference on Learning Representations (ICLR)*, 2019.

[CGK23] Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, et al. "Goat: Go to any thing." *arXiv preprint arXiv:2311.06430*, 2023.

[CHG16] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. "Guided search for task and motion plans using learned heuristics." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[CHY19] Yixin Chen, Siyuan Huang, Tao Yuan, Siyuan Qi, Yixin Zhu, and Song-Chun Zhu. "Holistic++ scene understanding: Single-view 3D holistic scene parsing and human pose estimation with human-object interaction and physical commonsense." In *International Conference on Computer Vision (ICCV)*, 2019.

[CSC14] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. "Reducing the barrier to entry of complex robotic software: a moveit! case study." *arXiv preprint arXiv:1404.3785*, 2014.

[CST22] Aidan Curtis, Tom Silver, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Kaelbling. "Discovering state and action abstractions for generalized task and motion planning." In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, pp. 5377–5384, 2022.

[CTJ21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. "Evaluating large language models trained on code." *arXiv preprint arXiv:2107.03374*, 2021.

[DAD18] Coline Devin, Pieter Abbeel, Trevor Darrell, and Sergey Levine. "Deep object-centric representations for generalizable robot learning." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[DCS17] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[DHT20] Danny Driess, Jung-Su Ha, and Marc Toussaint. "Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image." In *Robotics: Science and Systems (RSS)*, 2020.

[DKC16] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. "Incremental task and motion planning: A constraint-based approach." In *Robotics: Science and Systems (RSS)*, 2016.

[DKD23] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. "Vision-language models as success detectors." *arXiv preprint arXiv:2303.07280*, 2023.

[DLD22] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. "A survey for in-context learning." *arXiv:2301.00234*, 2022.

[DLL22] Yifan Du, Zikang Liu, Junyi Li, and Wayne Xin Zhao. "A survey of vision-language pre-trained models." *arXiv preprint arXiv:2202.10936*, 2022.

[DXM20] Xinke Deng, Yu Xiang, Arsalan Mousavian, Clemens Eppner, Timothy Bretl, and Dieter Fox. "Self-supervised 6d object pose estimation for robot manipulation." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[DZA23] Yan Ding, Xiaohan Zhang, Saeid Amiri, Nieqing Cao, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. "Integrating Action Knowledge and LLMs for Task Planning and Situation Handling in Open Worlds." *Autonomous Robots*, 2023.

[DZP23] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. "Task and motion planning with large language models for object rearrangement." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[EGL19] Mark Edmonds, Feng Gao, Hangxin Liu, Xu Xie, Siyuan Qi, Brandon Rothrock, Yixin Zhu, Ying Nian Wu, Hongjing Lu, and Song-Chun Zhu. "A tale of two explanations: Enhancing human trust by explaining robot behavior." *Science Robotics*, **4**(37), 2019.

[EGX17] Mark Edmonds, Feng Gao, Xu Xie, Hangxin Liu, Siyuan Qi, Yixin Zhu, Brandon Rothrock, and Song-Chun Zhu. "Feeling the force: Integrating force and pose for fluent discovery through imitation learning to open medicine bottles." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[FGE23] Xiaolin Fang, Caelan Reed Garrett, Clemens Eppner, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. "DiMSam: Diffusion Models as Samplers for Task and Motion Planning under Partial Observability." *arXiv:2306.13196*, 2023.

[FL03] Maria Fox and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains." *Journal of Artificial Intelligence Research*, **20**:61–124, 2003.

[FN71] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial intelligence*, **2**(3-4):189–208, 1971.

[FNF18] Fadri Furrer, Tonci Novkovic, Marius Fehr, Abel Gawel, Margarita Grinvald, Torsten Sattler, Roland Siegwart, and Juan Nieto. "Incremental object database: Building 3d models from multiple partial observations." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6835–6842. IEEE, 2018.

[FTT23] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. "Foundation models in robotics: Applications, challenges, and the future." *arXiv preprint arXiv:2312.07843*, 2023.

[GAG15] Saurabh Gupta, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. "Aligning 3D models to RGB-D images of cluttered scenes." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[Gat07] B Gates. "A robot in every home." *Scientific American magazine*, 2007.

[GCB23] Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. "Navigating to objects in the real world." *Science Robotics*, **8**(79):eadf6991, 2023.

[GCH21] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Integrated task and motion planning." *Annual review of control, robotics, and autonomous systems*, 2021.

[GCS22] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. "Multi-skill Mobile Manipulation for Object Rearrangement." In *International Conference on Learning Representations (ICLR)*, 2022.

[GFN19] Margarita Grinvald, Fadri Furrer, Tonci Novkovic, Jen Jen Chung, Cesar Cadena, Roland Siegwart, and Juan Nieto. "Volumetric instance-aware semantic mapping and 3D object discovery." *IEEE Robotics and Automation Letters (RA-L)*, **4**(3):3037–3044, 2019.

[GHM10] Tilbe Göksun, Kathy Hirsh-Pasek, and Roberta Michnick Golinkoff. "Trading spaces: Carving up events for learning language." *Perspectives on Psychological Science*, **5**(1):33–42, 2010.

[Gib50] James J Gibson. *The perception of the visual world.* Houghton Mifflin, 1950.

[Gib66] James J Gibson. *The senses considered as perceptual systems.* Houghton Mifflin, 1966.

[GKM23] Qiao Gu, Alihusein Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, et al. "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning." *arXiv preprint arXiv:2309.16650*, 2023.

[GLK18] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. "Ffrob: Leveraging symbolic planning for efficient task and motion planning." *International Journal of Robotics Research (IJRR)*, 2018.

[GLK20] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning." In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020.

[GN91] Naresh Gupta, Dana S Nau, et al. "Complexity Results for Blocks-World Planning." In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 91, pp. 629–633. Citeseer, 1991.

[GPL20] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. "Online replanning in belief space for partially observable task and motion problems." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5678–5684. IEEE, 2020.

[GWZ23] Yanjiang Guo, Yen-Jen Wang, Lihan Zha, Zheyuan Jiang, and Jianyu Chen. "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment." *arXiv preprint arXiv:2307.00329*, 2023.

[HAP22] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents." In *International Conference on Machine Learning (ICML)*, 2022.

[Hel06] Malte Helmert. "The fast downward planning system." *Journal of Artificial Intelligence Research*, **26**:191–246, 2006.

[HGD17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In *International Conference on Computer Vision (ICCV)*, 2017.

[HGM23] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. "Reasoning with language model is planning with world model." *arXiv preprint arXiv:2305.14992*, 2023.

[HLB19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. "Dream to Control: Learning Behaviors by Latent Imagination." In *International Conference on Learning Representations (ICLR)*, 2019.

[HLF19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels." In *International Conference on Machine Learning (ICML)*, 2019.

[HLF22] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. "Deep hierarchical planning from pixels." *Advances in Neural Information Processing Systems (NeurIPS)*, **35**:26091–26104, 2022.

[HLS19] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. "Chainqueen: A real-time differentiable physical simulator for soft robotics." In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[HLS20] Dinh-Cuong Hoang, Achim J Lilienthal, and Todor Stoyanov. "Panoptic 3D Mapping and Object Pose Estimation Using Adaptively Weighted Semantic Information." *IEEE Robotics and Automation Letters (RA-L)*, **5**(2):1962–1969, 2020.

[HLZ23] Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. "Look Before You Leap: Unveiling the Power of GPT-4V in Robotic Vision-Language Planning." *arXiv preprint arXiv:2311.17842*, 2023.

[HN11] Kris Hauser and Victor Ng-Thow-Hing. "Randomized multi-modal motion planning for a humanoid robot manipulation task." *International Journal of Robotics Research (IJRR)*, 2011.

[HNX19] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. "Neural task graphs: Generalizing to unseen tasks from a single video demonstration." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8565–8574, 2019.

[HPN16] Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. "Scenenn: A scene meshes dataset with annotations." In *International Conference on 3D Vision (3DV)*, 2016.

[HQX18] Siyuan Huang, Siyuan Qi, Yinxue Xiao, Yixin Zhu, Ying Nian Wu, and Song-Chun Zhu. "Cooperative holistic scene understanding: Unifying 3d object, layout, and camera pose estimation." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[HQZ18] Siyuan Huang, Siyuan Qi, Yixin Zhu, Yinxue Xiao, Yuanlu Xu, and Song-Chun Zhu. "Holistic 3d scene parsing and reconstruction from a single rgb image." In *European Conference on Computer Vision (ECCV)*, 2018.

[HTY18] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. "Pointwise convolutional neural networks." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[HWS22] Nicklas Hansen, Xiaolong Wang, and Hao Su. "Temporal Difference Learning for Model Predictive Control." In *International Conference on Machine Learning (ICML)*, 2022.

[HXX23] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. "Inner Monologue: Embodied Reasoning through Planning with Language Models." In *Conference on Robot Learning (CoRL)*, 2023.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[HZJ21] Muzhi Han, Zeyu Zhang, Ziyuan Jiao, Xu Xie, Yixin Zhu, Song-Chun Zhu, and Hangxin Liu. "Reconstructing interactive 3d scenes by panoptic mapping and cad model alignments." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12199–12206. IEEE, 2021.

[HZJ22] Muzhi Han, Zeyu Zhang, Ziyuan Jiao, Xu Xie, Yixin Zhu, Song-Chun Zhu, and Hangxin Liu. "Scene reconstruction with functional objects for robot autonomy." *International Journal of Computer Vision (IJCV)*, **130**(12):2940–2961, 2022.

[HZX20] Lei Han, Tian Zheng, Lan Xu, and Lu Fang. "Occuseg: Occupancy-aware 3d instance segmentation." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[HZZ24] Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. "InterPreT: Interactive Predicate Learning from Language Feedback for Generalizable Task Planning." In *Robotics: Science and Systems (RSS)*, 2024.

[IH92] Katsushi Ikeuchi and Martial Hebert. "Task-oriented vision." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1992.

[JCH20] Baoxiong Jia, Yixin Chen, Siyuan Huang, Yixin Zhu, and Song-Chun Zhu. "LEMMA: A Multi-view Dataset for LEarning Multi-agent Multi-task Activities." In *European Conference on Computer Vision (ECCV)*, 2020.

[JFZ19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to trust your model: Model-based policy optimization." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[JHZ22] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. "Ditto: Building digital twins of articulated objects from interaction." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[JLC21] Ajinkya Jain, Rudolf Lioutikov, Caleb Chuck, and Scott Niekum. "Screwnet: Category-independent articulation model estimation from depth images using screw theory." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[JLT13] Nikolay Jetchev, Tobias Lang, and Marc Toussaint. "Learning grounded relational symbols from continuous data for abstract reasoning." In *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*, 2013.

[JLU23] Vidhi Jain, Yixin Lin, Eric Undersander, Yonatan Bisk, and Akshara Rai. "Transformers are adaptable task planners." In *Conference on Robot Learning (CoRL)*, 2023.

[JNZ22] Ziyuan Jiao, Yida Niu, Zeyu Zhang, Song-Chun Zhu, Yixin Zhu, and Hangxin Liu. "Sequential Manipulation Planning on Scene Graph." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[JQZ18] Chenfanfu Jiang, Siyuan Qi, Yixin Zhu, Siyuan Huang, Jenny Lin, Lap-Fai Yu, Demetri Terzopoulos, and Song-Chun Zhu. "Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars." *International Journal of Computer Vision (IJCV)*, **126**(9):920–941, 2018.

[JRK21] Steven James, Benjamin Rosman, and George Konidaris. "Autonomous learning of object-centric abstractions for high-level planning." In *International Conference on Learning Representations (ICLR)*, 2021.

[JV87] Roy Jonker and Anton Volgenant. "A shortest augmenting path algorithm for dense and sparse linear assignment problems." *Computing*, **38**(4):325–340, 1987.

[JZJ21] Ziyuan Jiao, Zeyu Zhang, Xin Jiang, David Han, Song-Chun Zhu, Yixin Zhu, and Hangxin Liu. "Consolidating Kinematic Models to Promote Coordinated Mobile Manipulations." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[JZW21a] Ziyuan Jiao, Zhang Zeyu, Weiqi Wang, David Han, Song-Chun Zhu, Yixin Zhu, and Hangxin Liu. "Efficient Task Planning for Mobile Manipulation: a Virtual Kinematic Chain Perspective." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[JZW21b] Ziyuan Jiao, Zeyu Zhang, Weiqi Wang, David Han, Song-Chun Zhu, Yixin Zhu, and Hangxin Liu. "Efficient Task Planning for Mobile Manipulation: a Virtual Kinematic Chain Perspective." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[Kae20] Leslie Pack Kaelbling. "The foundation of efficient robot learning." *Science*, **369**(6506):915–916, 2020.

[KGR22] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. "Large language models are zero-shot reasoners." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. "Optimization by simulated annealing." *Science*, **220**(4598):671–680, 1983.

[Kha99] Oussama Khatib. "Mobile manipulation: The robotic assistant." *Robotics and Autonomous Systems*, **26**(2-3):175–183, 1999.

[KHC23] Amita Kamath, Jack Hessel, and Kai-Wei Chang. "What's" up" with vision-language models? Investigating their struggle with spatial reasoning." *arXiv preprint arXiv:2310.19785*, 2023.

[KHD18] Lars Kunze, Nick Hawes, Tom Duckett, Marc Hanheide, and Tomáš Krajník. "Artificial intelligence for long-term robot autonomy: A survey." *IEEE Robotics and Automation Letters (RA-L)*, **3**(4):4023–4030, 2018.

[KHG19] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. "Panoptic segmentation." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[KJ12] Brian Kulis and Michael I. Jordan. "Revisiting K-Means: New Algorithms via Bayesian Nonparametrics." In *International Conference on Machine Learning (ICML)*, 2012.

[KKL18a] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "From skills to symbols: Learning symbolic representations for abstract high-level planning." *Journal of Artificial Intelligence Research*, **61**:215–289, 2018.

[KKL18b] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "From skills to symbols: Learning symbolic representations for abstract high-level planning." *Journal of Machine Learning Research (JMLR)*, 2018.

[KL05] Sven Koenig and Maxim Likhachev. "Fast replanning for navigation in unknown terrain." *IEEE Transactions on Robotics (T-RO)*, **21**(3):354–363, 2005.

[KL11] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1470–1477. IEEE, 2011.

[KR96] David C Knill and Whitman Richards. *Perception as Bayesian inference*. Cambridge University Press, 1996.

[KS20] Beomjoon Kim and Luke Shimanuki. "Learning value functions with relational state representations for guiding task-and-motion planning." In *Conference on Robot Learning (CoRL)*, 2020.

[Kuh55] Harold W Kuhn. "The Hungarian method for the assignment problem." *Naval Research Logistics Quarterly*, **2**(1-2):83–97, 1955.

[KZG17] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." *International Journal of Computer Vision (IJCV)*, **123**:32–73, 2017.

[LaV98] Steven M LaValle. "Rapidly-exploring random trees: A new tool for path planning." Technical report, Computer Science Dept., Iowa State Univ., Ames, IA, 1998. Unpublished.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[LBS23] Zeyi Liu, Arpit Bahety, and Shuran Song. "REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction." In *Conference on Robot Learning (CoRL)*, pp. 3468–3484. PMLR, 2023.

[LCZ23] Huihan Liu, Alice Chen, Yuke Zhu, Adith Swaminathan, Andrey Kolobov, and Ching-An Cheng. "Interactive Robot Learning from Verbal Correction." *arXiv preprint arXiv:2310.17555*, 2023.

[LHL22] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B. Tenenbaum, David Held, and Chuang Gan. "DiffSkill: Skill Abstraction from Differentiable Physics for Deformable Object Manipulations with Tools." In *International Conference on Learning Representations (ICLR)*, 2022.

[LHX23] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. "Code as policies: Language model programs for embodied control." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

[LJZ23] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. "Llm+ p: Empowering large language models with optimal planning proficiency." *arXiv preprint arXiv:2304.11477*, 2023.

[LLK19] Xueting Li, Sifei Liu, Kihwan Kim, Xiaolong Wang, Ming-Hsuan Yang, and Jan Kautz. "Putting humans in a scene: Learning affordance in 3d indoor environments." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[LM19] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation." *Journal of Field Robotics*, **36**(2):416–446, 2019.

[LMB14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In *European Conference on Computer Vision (ECCV)*, 2014.

[LOP24] Peiqi Liu, Yaswanth Orru, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. "Ok-robot: What really matters in integrating open-knowledge models for robotics." *arXiv preprint arXiv:2401.12202*, 2024.

[LPP22] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. "Pre-trained language models for interactive decision-making." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[LS23] Amber Li and Tom Silver. "Embodied Active Learning of Relational State Abstractions for Bilevel Planning." In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023.

[LSA19] João Loula, Tom Silver, Kelsey R Allen, and Josh Tenenbaum. "Discovering a symbolic planning language from continuous experience." In *Annual Meeting of the Cognitive Science Society (CogSci)*, p. 2193, 2019.

[LXS18] Ligang Liu, Xi Xia, Han Sun, Qi Shen, Juzhan Xu, Bin Chen, Hui Huang, and Kai Xu. "Object-aware guidance for autonomous scene reconstruction." *ACM Transactions on Graphics (TOG)*, **37**(4):1–12, 2018.

[LZS18] Hangxin Liu, Yaofang Zhang, Wenwen Si, Xu Xie, Yixin Zhu, and Song-Chun Zhu. "Interactive robot knowledge patching using augmented reality." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[LZZ19] Hangxin Liu, Chi Zhang, Yixin Zhu, Chenfanfu Jiang, and Song-Chun Zhu. "Mirroring without overimitation: Learning functionally equivalent manipulation actions." In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[Man92] Jean M Mandler. "How to build a baby: II. Conceptual primitives." *Psychological review*, **99**(4):587, 1992.

[MB02] Grégoire Malandain and Jean-Daniel Boissonnat. "Computing the diameter of a point set." *International Journal of Computational Geometry & Applications*, **12**(06):489–509, 2002.

[MB19] Roberto Martín-Martín and Oliver Brock. "Coupled recursive estimation for online interactive perception of articulated objects." *International Journal of Robotics Research (IJRR)*, pp. 1–37, 2019.

[MB22] Toki Migimatsu and Jeannette Bohg. "Grounding predicates through actions." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3498–3504. IEEE, 2022.

[MCB18] John McCormac, Ronald Clark, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. "Fusion++: Volumetric object-level slam." In *International Conference on 3D Vision (3DV)*, 2018.

[MFM04] David R Martin, Charless C Fowlkes, and Jitendra Malik. "Learning to detect natural image boundaries using local brightness, color, and texture cues." *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, **26**(5):530–549, 2004.

[MGF22] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. "kpam: Keypoint affordances for category-level robotic manipulation." In *International Symposium on Robotics Research (ISRR)*, 2022.

[MGK19] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. "The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision." *arXiv preprint arXiv:1904.12584*, 2019.

[MHD17] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[MJP92] Steven Minton, Mark D Johnston, Andrew B Philips, and Philip Laird. "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems." *Artificial Intelligence*, **58**(1-3):161–205, 1992.

[MLT22] Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. "PDSketch: Integrated Domain Programming, Learning, and Planning." *Advances in Neural Information Processing Systems (NeurIPS)*, **35**:36972–36984, 2022.

[MLW23] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. "Eureka: Human-level reward design via coding large language models." *arXiv preprint arXiv:2310.12931*, 2023.

[Mor78] Jorge J Moré. "The Levenberg-Marquardt algorithm: implementation and theory." In *Numerical Analysis*, pp. 105–116. Springer, 1978.

[MT17] Raul Mur-Artal and Juan D Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras." *IEEE Transactions on Robotics (T-RO)*, **33**(5):1255–1262, 2017.

[MTF15] Austin Myers, Ching L Teo, Cornelia Fermüller, and Yiannis Aloimonos. "Affordance detection of tool parts from geometric features." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[MXF23] Suvir Mirchandani, Fei Xia, Pete Florence, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, Andy Zeng, et al. "Large Language Models as General Pattern Machines." In *Conference on Robot Learning (CoRL)*, 2023.

[MZC19] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. "Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[New36] Isaac Newton. *The Method of Fluxions and Infinite Series: With Its Application to the Geometry of Curve Lines*. Nourse, 1736.

[Nil84] Nils J Nilsson et al. *Shakey the robot*, volume 323. Sri International Menlo Park, California, 1984.

[NMB21] Michael Noseworthy, Caris Moses, Isaiah Brand, Sebastian Castro, Leslie Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. "Active learning of abstract plan feasibility." *arXiv:2107.00683*, 2021.

[NO00] Ulrich Nehmzow and Carl Owen. "Robot navigation in the real world:: Experiments with Manchester's FortyTwo in unmodified, large environments." *Robotics and Autonomous systems*, **33**(4):223–242, 2000.

[NSI19] Gaku Narita, Takashi Seno, Tomoya Ishikawa, and Yohsuke Kaji. "PanopticFusion: Online Volumetric Semantic Mapping at the Level of Stuff and Things." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[Ope] OpenAI. "ChatGPT." `https://chat.openai.com/`.

[Ope23a] OpenAI. "GPT-4 Technical Report." *arXiv preprint arXiv:2303.08774*, 2023.

[Ope23b] OpenAI. "GPT-4V(ision) System Card." `https://cdn.openai.com/papers/GPTV_System_Card.pdf`, 2023.

[OTF17] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[PHN19] Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. "Real-time progressive 3D semantic segmentation for indoor scenes." In *Proceedings of Winter Conference on Applications of Computer Vision (WACV)*, 2019.

[PJ12] Andrzej Pronobis and Patric Jensfelt. "Large-scale semantic mapping and reasoning with heterogeneous modalities." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[PJT02] Lars Petersson, Patric Jensfelt, Dennis Tell, Morten Strandberg, Danica Kragic, and Henrik I Christensen. "Systems integration for real-world manipulation tasks." In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pp. 2500–2505. IEEE, 2002.

[PNH19] Quang-Hieu Pham, Thanh Nguyen, Binh-Son Hua, Gemma Roig, and Sai-Kit Yeung. "JSIS3D: joint semantic-instance segmentation of 3d point clouds with multi-task pointwise networks and multi-value conditional random fields." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[PNZ17] Seyoung Park, Bruce Xiaohan Nie, and Song-Chun Zhu. "Attribute and-or grammar for joint parsing of human pose, parts and attributes." *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, **40**(7):1555–1569, 2017.

[PTL18] Quang-Hieu Pham, Minh-Khoi Tran, Wenhui Li, Shu Xiang, Heyu Zhou, Weizhi Nie, Anan Liu, Yuting Su, Minh-Triet Tran, Ngoc-Minh Bui, et al. "SHREC'18: RGB-D object-to-CAD retrieval." In *3DOR: Proceedings of the 11th Eurographics Workshop on 3D Object Retrieval*, 2018.

[Put90] Martin L Puterman. "Markov decision processes." *Handbooks in operations research and management science*, **2**:331–434, 1990.

[PZK07] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. "Learning symbolic models of stochastic domains." *Journal of Artificial Intelligence Research*, **29**:309–352, 2007.

[QCG09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. "ROS: an open-source Robot Operating System." In *ICRA workshop on open source software*, volume 3, p. 5. Kobe, Japan, 2009.

[QJH20] Siyuan Qi, Baoxiong Jia, Siyuan Huang, Ping Wei, and Song-Chun Zhu. "A Generalized Earley Parser for Human Activity Parsing and Prediction." *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[QYS17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[QZH18] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. "Human-centric indoor scene synthesis using stochastic grammar." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[RGA20] Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. "3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans." In *Robotics: Science and Systems (RSS)*, 2020.

[RHG16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: towards real-time object detection with region proposal networks." *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, **39**(6):1137–1149, 2016.

[RJO23] Eric Rosen, Steven James, Sergio Orozco, Vedant Gupta, Max Merlin, Stefanie Tellex, and George Konidaris. "Synthesizing Navigation Abstractions for Planning with Portable Manipulation Skills." In *Conference on Robot Learning (CoRL)*, pp. 2278–2287. PMLR, 2023.

[RKH21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision." In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

[RLZ24] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. "Grounded sam: Assembling open-world models for diverse visual tasks." *arXiv preprint arXiv:2401.14159*, 2024.

[Rus10] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[SBM23] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. "Progprompt: Generating situated robot task plans using large language models." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

[SCB23] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. "Reflexion: Language Agents with Verbal Reinforcement Learning.", 2023.

[SCK23] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. "Predicate invention for bilevel planning." In *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[SCT21] Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Learning symbolic operators for task and motion planning." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[SCX20] Zhiqiang Sui, Haonan Chang, Ning Xu, and Odest Chadwicke Jenkins. "Geofusion: geometric consistency informed scene estimation in dense clutter." *IEEE Robotics and Automation Letters (RA-L)*, **5**(4):5913–5920, 2020.

[Set96] James A Sethian. "A fast marching level set method for monotonically advancing fronts." *proceedings of the National Academy of Sciences*, **93**(4):1591–1595, 1996.

[SFR14] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[SHK12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. "Indoor segmentation and support inference from rgbd images." In *European Conference on Computer Vision (ECCV)*. Springer, 2012.

[SHL22] Jiankai Sun, De-An Huang, Bo Lu, Yun-Hui Liu, Bolei Zhou, and Animesh Garg. "PlaTe: Visually-grounded planning with transformers in procedural tasks." *IEEE Robotics and Automation Letters (RA-L)*, **7**(2):4924–4930, 2022.

[SHM16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature*, **529**(7587):484–489, 2016.

[SHS22] Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "Pddl planning with pretrained large language models." In *NeurIPS 2022 foundation models for decision making workshop*, 2022.

[SLJ23] Yao Su, Jiarui Li, Ziyuan Jiao, Meng Wang, Chi Chu, Hang Li, Yixin Zhu, and Hangxin Liu. "Sequential Manipulation Planning for Over-actuated Unmanned Aerial Manipulators." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[SMT21] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[SSB11] Jürgen Sturm, Cyrill Stachniss, and Wolfram Burgard. "A probabilistic framework for learning kinematic models of articulated objects." *Journal of Artificial Intelligence Research*, **41**:477–526, 2011.

[SWW23] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. "Llm-planner: Few-shot grounded planning for embodied agents with large language models." In *International Conference on Computer Vision (ICCV)*, pp. 2998–3009, 2023.

[SZY24] Marta Skreta, Zihan Zhou, Jia Lin Yuan, Kourosh Darvish, Alán Aspuru-Guzik, and Animesh Garg. "Replan: Robotic replanning with perception and language models." *arXiv preprint arXiv:2401.04157*, 2024.

[TAS18] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. "Differentiable physics and stable modes for tool-use and manipulation planning." In *Robotics: Science and Systems (RSS)*, 2018.

[TJR13] Yuichi Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng. "Point-plane SLAM for hand-held 3D sensors." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[Tou15] Marc Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[UAR21] Elena Umili, Emanuele Antonioni, Francesco Riccio, Roberto Capobianco, Daniele Nardi, and Giuseppe De Giacomo. "Learning a symbolic planning domain through the interaction with continuous environments." In *ICAPS PRL Workshop*, 2021.

[UP15] Emre Ugur and Justus Piater. "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2627–2633. IEEE, 2015.

[VCC20] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. "Entity abstraction in visual model-based reinforcement learning." In *Conference on Robot Learning (CoRL)*, 2020.

[VOS22] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. "Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change)." *arXiv preprint arXiv:2206.10498*, 2022.

[VRS21] Mel Vecerik, Jean-Baptiste Regli, Oleg Sushkov, David Barker, Rugile Pevceviciute, Thomas Rothörl, Raia Hadsell, Lourdes Agapito, and Jonathan Scholz. "S3k: Self-supervised semantic keypoints for robotic manipulation via multi-view consistency." In *Conference on Robot Learning (CoRL)*, 2021.

[WB21] Bowen Wen and Kostas Bekris. "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[WCC23] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. "Describe, explain, plan and select: interactive planning with LLMs enables open-world multi-task agents." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[WCL23] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. "Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models." *arXiv preprint arXiv:2311.05997*, 2023.

[WDN20] Johanna Wald, Helisa Dhamo, Nassir Navab, and Federico Tombari. "Learning 3D Semantic Scene Graphs from 3D Indoor Reconstructions." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[WDS19] Andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. "Learning feasibility for task and motion planning in tabletop environments." *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[WGK18] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Active model learning and diverse action sampling for task and motion planning." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[WHJ24] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. "LLM3:Large Language Model-based Task and Motion Planning with Motion Failure Reasoning." *arXiv preprint arXiv:2403.11552*, 2024.

[WKM19] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. "Detectron2." `https://github.com/facebookresearch/detectron2`, 2019.

[WMR10] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. "Combined task and motion planning for mobile manipulation." In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, pp. 254–257, 2010.

[WSJ20] Kentaro Wada, Edgar Sucar, Stephen James, Daniel Lenton, and Andrew J Davison. "MoreFusion: Multi-object Reasoning for 6D Pose Estimation from Volumetric Fusion." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[WWS22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. "Chain-of-thought prompting elicits reasoning in large language models." *Advances in Neural Information Processing Systems (NeurIPS)*, **35**:24824–24837, 2022.

[WWX23] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. "Embodied task planning with large language models." *arXiv preprint arXiv:2307.01848*, 2023.

[WWZ21] Yijia Weng, He Wang, Qiang Zhou, Yuzhe Qin, Yueqi Duan, Qingnan Fan, Baoquan Chen, Hao Su, and Leonidas J Guibas. "Captra: Category-level pose tracking for rigid and articulated objects from point clouds." In *International Conference on Computer Vision (ICCV)*, 2021.

[WXJ23] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*, 2023.

[XHS15] Kai Xu, Hui Huang, Yifei Shi, Hao Li, Pinxin Long, Jianong Caichen, Wei Sun, and Baoquan Chen. "Autoscanning for coupled scene reconstruction and proactive object analysis." *ACM Transactions on Graphics (TOG)*, **34**(6):1–14, 2015.

[XLZ19] Xu Xie, Hangxin Liu, Zhenliang Zhang, Yuxing Qiu, Feng Gao, Siyuan Qi, Yixin Zhu, and Song-Chun Zhu. "Vrgym: A virtual testbed for physical and interactive ai." In *Proceedings of the ACM Turing Celebration Conference-China*, pp. 1–6, 2019.

[XMH19] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li F Fei-Fei. "Regression planning networks." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[XNZ18] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. "Neural task programming: Learning to generalize across hierarchical tasks." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3795–3802. IEEE, 2018.

[XYZ23] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. "Translating natural language to planning goals with large-language models." *arXiv preprint arXiv:2302.05128*, 2023.

[XZC17] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. "Scene graph generation by iterative message passing." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5410–5419, 2017.

[Yam97] Brian Yamauchi. "A frontier-based approach for autonomous exploration." In *International Symposium on Computational Intelligence in Robotics and Automation'*, pp. 146–151. IEEE, 1997.

[YCT23] Naoki Yokoyama, Alex Clegg, Joanne Truong, Eric Undersander, Tsung-Yen Yang, Sergio Arnaud, Sehoon Ha, Dhruv Batra, and Akshara Rai. "ASC: Adaptive Skill Coordination for Robotic Mobile Manipulation." *IEEE Robotics and Automation Letters (RA-L)*, **9**(1):779–786, 2023.

[YGF23] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. "Language to Rewards for Robotic Skill Synthesis." *arXiv preprint arXiv:2306.08647*, 2023.

[YGL23] Zhutian Yang, Caelan Garrett, Tomas Lozano-Perez, Leslie Kaelbling, and Dieter Fox. "Sequence-Based Plan Feasibility Prediction for Efficient Task and Motion Planning." In *Robotics: Science and Systems (RSS)*, 2023.

[YLF20] Tao Yuan, Hangxin Liu, Lifeng Fan, Zilong Zheng, Tao Gao, Yixin Zhu, and Song-Chun Zhu. "Joint Inference of States, Robot Knowledge, and Human (False-)Beliefs." In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[YRY23] Sriram Yenamandra, Arun Ramachandran, Karmesh Yadav, Austin Wang, Mukul Khanna, Theophile Gervet, Tsung-Yen Yang, Vidhi Jain, Alexander William Clegg, John Turner, et al. "Homerobot: Open-vocabulary mobile manipulation." *arXiv preprint arXiv:2306.11565*, 2023.

[YS19a] Shichao Yang and Sebastian Scherer. "Cubeslam: Monocular 3-d object slam." *IEEE Transactions on Robotics (T-RO)*, **35**(4):925–938, 2019.

[YS19b] Shichao Yang and Sebastian Scherer. "Monocular object and plane SLAM in structured environments." *IEEE Robotics and Automation Letters (RA-L)*, **4**(4):3145–3152, 2019.

[YZW19] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. "Gspn: Generative shape proposal network for 3d instance segmentation in point cloud." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[YZY22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. "ReAct: Synergizing Reasoning and Acting in Language Models." In *International Conference on Learning Representations (ICLR)*, 2022.

[ZDA23] Xiaohan Zhang, Yan Ding, Saeid Amiri, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. "Grounding Classical Task Planners via Vision-Language Models." *arXiv preprint arXiv:2304.08587*, 2023.

[ZGF20] Yixin Zhu, Tao Gao, Lifeng Fan, Siyuan Huang, Mark Edmonds, Hangxin Liu, Feng Gao, Chi Zhang, Siyuan Qi, Ying Nian Wu, et al. "Dark, beyond deep: A paradigm shift to cognitive ai with humanlike common sense." *Engineering*, **6**(3):310–345, 2020.

[ZGL19] Chuhang Zou, Ruiqi Guo, Zhizhong Li, and Derek Hoiem. "Complete 3D scene parsing from an RGBD image." *International Journal of Computer Vision (IJCV)*, **127**(2):143–162, 2019.

[ZHJ23] Zeyu Zhang, Muzhi Han, Baoxiong Jia, Ziyuan Jiao, Yixin Zhu, Song-Chun Zhu, and Hangxin Liu. "Learning a Causal Transition Model for Object Cutting." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1996–2003. IEEE, 2023.

[ZJZ16] Yixin Zhu, Chenfanfu Jiang, Yibiao Zhao, Demetri Terzopoulos, and Song-Chun Zhu. "Inferring forces and learning human utilities from videos." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[ZM07] Song-Chun Zhu and David Mumford. "A Stochastic Grammar of Images." *Foundations and Trends in Computer Graphics and Vision*, **2**(4):259–362, 2007.

[ZMK17] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3357–3364. IEEE, 2017.

[ZS89] Kaizhong Zhang and Dennis Shasha. "Simple fast algorithms for the editing distance between trees and related problems." *SIAM journal on computing*, **18**(6):1245–1262, 1989.

[ZTB21] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs." In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6541–6548. IEEE, 2021.

[ZZ11] Yibiao Zhao and Song-Chun Zhu. "Image parsing with stochastic scene grammar." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[ZZ13] Yibiao Zhao and Song-Chun Zhu. "Scene parsing by integrating function, geometry and appearance models." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[ZZC19] Jiaying Zhang, Xiaoli Zhao, Zheng Chen, and Zhejun Lu. "A review of deep learning-based semantic segmentation for point cloud." *IEEE Access*, **7**:179118–179133, 2019.

[ZZH24] Peiyuan Zhi, Zhiyuan Zhang, Muzhi Han, Zeyu Zhang, Zhitian Li, Ziyuan Jiao, Baoxiong Jia, and Siyuan Huang. "Closed-Loop Open-Vocabulary Mobile Manipulation with GPT-4V." *arXiv preprint arXiv:2404.10220*, 2024.

[ZZL23] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. "A survey of large language models." *arXiv preprint arXiv:2303.18223*, 2023.

[ZZW23] Zeyu Zhang, Lexing Zhang, Zaijin Wang, Ziyuan Jiao, Muzhi Han, Yixin Zhu, Song-Chun Zhu, and Hangxin Liu. "Part-level Scene Reconstruction Affords Robot Interaction." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[ZZY15] Bo Zheng, Yibiao Zhao, Joey Yu, Katsushi Ikeuchi, and Song-Chun Zhu. "Scene understanding by reasoning stability and safety." *International Journal of Computer Vision (IJCV)*, **112**(2):221–238, 2015.

[ZZZ15] Yixin Zhu, Yibiao Zhao, and Song-Chun Zhu. "Understanding tools: Task-oriented object modeling, learning and recognition." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[ZZZ20] Zhenliang Zhang, Yixin Zhu, and Song-Chun Zhu. "Graph-based Hierarchical Knowledge Representation for Robot Task Transfer from Virtual to Physical World." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.