SYMBOL GROUNDING FOR TASK AND MOTION PLANNING IN ROBOTICS

Xiaohan Zhang

BE, Renmin University of China, 2019
MS, State University of New York at Binghamton, 2020

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Ph.D. in Computer Science
in the Graduate School of the
State University of New York at Binghamton
2024

Accepted in partial fulfillment of the requirements for
the degree of Ph.D. in Computer Science
in the Graduate School of the
State University of New York at Binghamton
2024

Shiqi Zhang, Chair
Department of Computer Science, SUNY Binghamton

Lijun Yin, Member
Department of Computer Science, SUNY Binghamton

Jayson Boubin, Member
Department of Computer Science, SUNY Binghamton

Mohan Sridharan, Member
School of Informatics, University of Edinburgh (UK)

Sangwon Yoon, Outside Examiner
Department of Systems Science and Industrial Engineering, SUNY Binghamton

Abstract

For modern robots that are equipped with a set of skills, such as manipulation and navigation, it's crucial that they can autonomously make decisions to complete tasks over extended periods of time. To this end, researchers have been developing *planning* algorithms that allow robots to effectively sequence and use their skills to reach specific task-level goals. Existing planning systems have a strong reliance on structured and predictive world models which are often represented as symbol tokens. *Symbol grounding* is thus required to establish a meaningful connection between these abstract symbols and their real-world interpretations that robots can understand. A fundamental approach to symbol grounding for robots involves utilizing their perception capabilities, allowing the mapping of sensory readings to symbol tokens and thereby assigning perceptual meanings to symbols. Additionally, robots frequently need to plan their actions in the process of grounding, given robot actions being tightly coupled with perception. Motivated by the above observations, this dissertation focuses on 1) symbol grounding using perception and action, and 2) robot planning with grounded symbols. Our research draws on theories and methods from computer vision and machine learning with both being integrated with AI planning and continuous motion planning towards building long-horizon robot autonomy.

Acknowledgements

I want to express my deepest gratitude to my advisor, Prof. Shiqi Zhang. I still remember arriving in Binghamton without any research experience, and it was Shiqi who taught me from scratch how to conduct good research and how to become a responsible researcher. We spent many nights revising papers and brainstorming ideas, and these moments will remain unforgettable and invaluable throughout my life.

My PhD committee members, including Prof. Lijun Yin, Prof. Jayson Boubin, and Prof. Mohan Sridharan, have provided me with numerous insightful suggestions on my thesis and presentations. This dissertation would not have been possible without their help and careful revisions.

I also extend my gratitude to my collaborators from academia, including Prof. Jivko Sinapov from Tufts University, Prof. Peter Stone, Prof. Yuke Zhu, and Yifeng Zhu from UT Austin. The collaboration with Jivko led to my very first conference paper, which later became the first half of this dissertation, and I truly appreciate his suggestions along the way. Peter has been an excellent advisor, coauthoring every paper included in this dissertation and being a gracious host during my time as a visiting researcher at UT Austin. Yuke and his student Yifeng (co-advised by Peter) provided me with lots of precious ideas that made the second half of this dissertation possible.

I have been fortunate to spend time at Google Brain Robotics as a student researcher, and I want to thank my manager Montserrat Gonzalez Arenas, who taught me how to do good engineering in research by involving me in the exciting robotic table wiping project. I also spent time at Meta FAIR Labs with the fantastic Chris Paxton. I still cannot forget the time he was welcoming a new baby and helping me debug at the same time.

I will miss the time spent with my lab mates, especially the PhD students Saeid Amiri, Kishan Chandan, Yan Ding, Yohei Hayamizu, David Defazio, and Zainab Altaweel. We went to the gym, had parties, discussed research after a few drinks, and hiked on sunny days. Those were wonderful moments that helped me overcome many challenges.

Finally, I want to thank my dear family members. My mom, dad, and grandparents have witnessed my growth and given me the courage to get through the PhD journey. My lovely two dogs, Abigail and Mili, have been with me since I became a PhD student, and they are the most adorable creatures I have ever seen. Kexin Li, I can proudly say that we have known each other for half of my life. It is my great honor to have you by my side, giving me support, both emotionally and financially (as you know, PhD students are not among the wealthiest). Thank you, for being there, as always.

Contents

List of Tables

List of Figures

1 Introduction

In recent years, service robots are playing an increasingly important role, taking on complex, long-horizon tasks such as searching for objects, setting up tables, loading dishwashers, and organizing bookshelves. For robots that are equipped with a set of skills (e.g., manipulation and navigation), it's crucial that they can make decisions for task completion over extended periods of time. To achieve this, robotics researchers have been developing *planning algorithms* that support robots to effectively sequence and use their skills to reach specific goals [1].

Existing planning systems have a strong reliance on structured and predictive world models, enabling robots to reason about how the world is represented and functions so as to make accurate decisions. Symbolic representations are widely used in modeling world states and transitions [2]. Consider a robot given a task of "*Moving the red and empty coke can from the table to the chair.*" Symbol tokens such as `red` and `empty` can represent perceivable attributes of the coke can that the robot searches for [3]. `near(robot, chair)` can represent a robot physically being at a state that is close to the table [4, 5] and getting ready for picking up the target coke can. In reality, such world states, as opposed to a set of discrete symbols, are usually perceived through sensors while robots are operating in their continuous workspaces. Hence, a significant challenge persists in estab-

lishing a meaningful connection between these abstract symbols and their real-world interpretations that robots can understand, which is commonly known as *the symbol grounding problem* [6]. In this dissertation, we aim to address the symbol grounding problem by developing novel frameworks and algorithms that bridge the gap between symbolic representations and the continuous real-world observations made by robots. Furthermore, we utilize grounded symbols for advancing the field of robotic planning, making planning systems more adaptable, robust, and capable in unstructured and dynamic environments.

Breaking down the grounding problem and its robotic application, the first topic of this dissertation purely focuses on **symbol grounding** for robots, answering the question of:

How can robots ground symbols through perception and actions?

This grounding process is challenging because the robot has to first overcome imperfect perception due to sensor noise and partial observability [7]. This challenge usually persists in grounding symbols that have rich perceptual meanings such as attributes of objects. As an example, the `empty` attribute of an opaque can will always remain partially observable to the robot. Thus, robots frequently need to use their **actions** in the process of grounding, given robot actions being tightly coupled with perception [8]. For the purpose of identifying the `empty` symbol, the robot also needs to decide which of its many exploratory actions to perform on an object in order to effectively identify one specific symbol. From a human perspective, it is preferred to *shake* the object to figure out if it is `empty` according to the multimodal sensory information (e.g., sound) it produces, instead of taking a *poke* action which is less informative.

Grounding symbols that particularly have spatial meanings frequently requires mo-

bile manipulation capabilities. Consider an approach for grounding the `near(robot, table)` symbol, the robot needs to first take a navigation action to the table for estimating the distance from itself to the goal (e.g., the Euclidean distance between two points in the occupancy grid map) while accounting for unpredictable navigation noise. Then the robot may want to make sure the robot base is close enough by attempting to grasp an object on the table, such that `near(robot, table)` also becomes meaningful for successful manipulation [9, 10]. As the robot operates in cluttered environments, non-deterministic action outcomes at the motion level introduce an additional dimension of complexity to the above grounding process [11]. Navigation actions frequently lead to the robot being at slightly different locations from the exact given goals, and manipulation actions such as grasping is also an unsolved problem that many robotic researchers have been investigated for decades.

The second topic of this dissertation is on applying grounded symbols to robotic planning. Specifically, we investigate:

> *How grounded symbols facilitate robot task and motion planning for everyday tasks?*

and refer it as **grounded planning** in the dissertation. Task and motion planning (TAMP) is one type of the planning algorithms that have been used for planning at both symbolic (discrete) and geometric (continuous) levels, but it is not grounded in the sense that TAMP assumes perfect perception and full knowledge about how the world functions. This dissertation investigates a grounded version of TAMP that shares the same goal (i.e., computing feasible plans for service robots) with original TAMP algorithms, but with visually

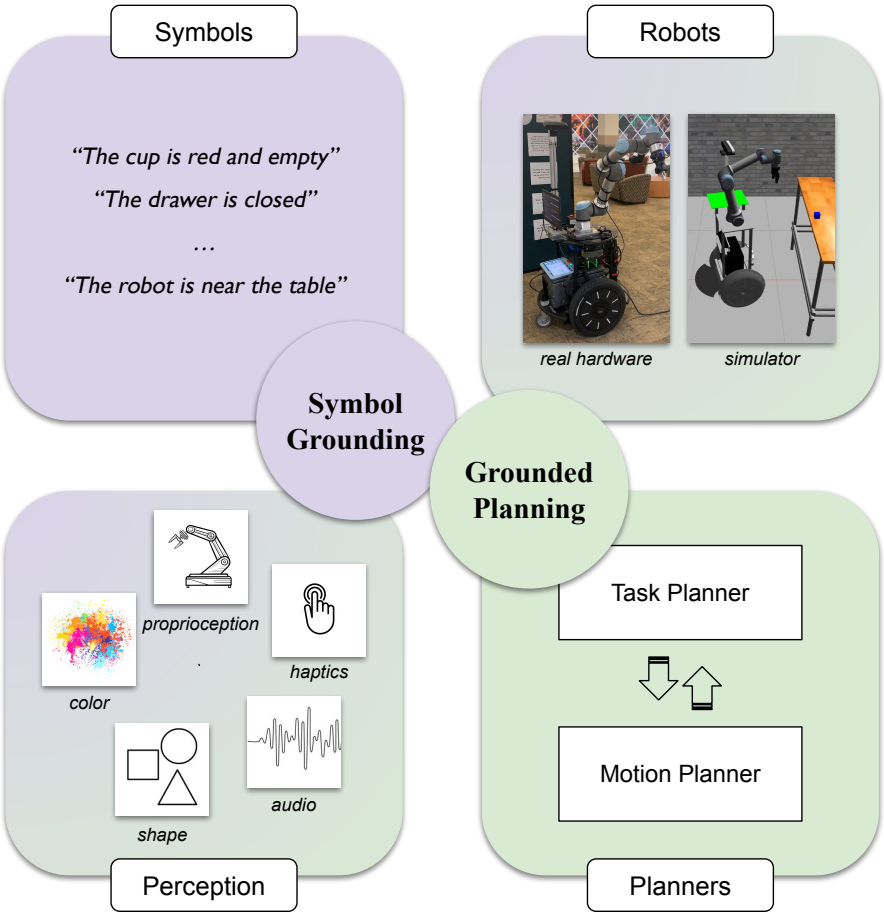


Figure 1.1: This dissertation focuses on symbol grounding and its robotic application on planning. Robots can use their action and perception capabilities to ground symbols, and grounded symbols can in turn benefit task and motion planning for completing long-horizon tasks.

grounded symbols in the task planner thereby removing the hard constraints of perfect perception. Operating in larger domains further require robot actions that take relatively long time (e.g., minutes or even hours as navigating from one location to another), where it is advantageous for TAMP algorithms to incorporate both plan efficiency and action feasibility into the evaluation of plan qualities. However, without the previous strong assumptions of deterministic action executions (made by classical planning), it will be much more challenging to achieve the optimality for grounded task and motion planning.

Table 1.1: Types of symbol grounding and the applied planning frameworks being used in each chapter.

Chapters	Types of Symbols	Types of Grounding	Planning Frameworks
4	object attributes	interactive	partially observable Markov decision process
5	spatial relationships	data-driven	task and motion planning
6	abstracted locations	data-driven	task and motion planning

To address the above challenges in symbol grounding and grounded planning, this dissertation demonstrates our research drawing on theories and methods from computer vision and machine learning with both being integrated with symbolic task planning and sampling-based motion planning. Specifically, related research will be discussed in the following chapters as: 1) Grounding object attributes through interactive perception (Chapter 3); 2) Visually grounded task and motion planning (Chapter 4); and 3) Optimizing task planners with grounded symbolic state space (Chapter 5). As summarized in Table 1.1, Chapter 3 focuses on grounding objects' attribute symbols with their perceivable meanings. Chapter 4 focuses on grounding task planner's symbolic predicates (i.e., spatial relationships such as `near`), and applying them for optimizing task-motion plans. Finally, Chapter 5 looks into constructing and grounding location symbols at the same time for optimizing task planners' state spaces.

1.1 Thesis Outline

The rest of this dissertation is structured into four chapters. Chapter 2 is a discussion of important building blocks related to this study, including automated planning (i.e., task planning), motion planning, and mobile manipulation. It is expected that this background section will aid readers in better understanding the subsequent parts of the dissertation.

Chapter 3 discusses how to interact with objects and ground object attribute symbols through exploratory actions. The robot task is to identify symbols such as `red` and `empty`, and the grounding process is "interactive", meaning the robot takes actions and grounds symbols at the same time. In this work, we adapt partially observable Markov decision processes (POMDPs) to sequence robot actions. The challenge for this interactive grounding process is that it requires the robot to learn an accurate and efficient action policy over the long term without pre-collected data. Attribute symbols from Chapter 3 are world representations purely perceivable at the task level, so all the low-level action primitives being used to ground such symbols can be predefined.

Chapter 4 discusses how to ground symbols that touch both task level and motion level. The proposed algorithm visually grounds spatial relationships (i.e., `near(robot, table)`), also known as the state-mapping function, to probabilistically evaluate action feasibility, and is particularly suitable for large domains with long-term robot operations (e.g., long-distance navigation). A "data-driven" approach is applied here for symbol grounding, where we collected a dataset in simulation that includes robot's past experience on conducting mobile manipulation tasks where in each data instance, a robot unloads an object with dynamic obstacles surrounding a table. The symbols of spatial relationships are then grounded into a form of action heatmaps generated from Fully Convolutional Neural Networks (FCNs) [12] that is trained on the collected data.

Location symbols from the above chapters are always predefined by a domain expert who manually specifies a symbolic state space in the task planner. Nevertheless, manually constructing state spaces might not be desirable in some scenarios. For instance, if each object is placed at a separate symbolic location as defined in the task-level state space, the

robot will always need to navigate before picking up the next object. This is because the task planner believes only a navigation action can bring the robot to the location required by the next manipulation action. In practice, however, the robot often picks up multiple objects from a single position, for example, as restaurant waiters can easily identify a standing location that allows them to pick up multiple dishes at once. Especially when objects are located close to each other, it is unnecessary for the robot to navigate before every manipulation. This observation motivates the development of Chapter 5 on optimizing task-level state spaces with grounded symbolic locations for task planners to generate feasible and efficient task-motion plans.

Finally, this dissertation concludes with a summary and an outlook on future work in Chapter 6.

1.2 Other Doctoral Research

Apart from the works that are included in this dissertation [4, 3, 5], I have also spent time conducting other research on robotic reasoning, planning and learning, briefly summarized here.

GHAL360 is a human-involved learning framework that enables the Mobile Telepresence Robots to learn a goal-oriented policy from reinforcements for guiding human attention using visual indicators [13]. We have also used RL in robotic table wiping tasks where we learned a policy using RL in simulation with a whole-body trajectory optimization framework to realize the wiping action execution in real world [14]. One of our recent works called SLAP uses imitation learning taking three-dimensional tokens as the input representation to train a single multi-task, language-conditioned action prediction

policy [15]. For planning research, TMPUD is one of the very first frameworks that applies task and motion planning in autonomous driving domains [16]. We then developed Task-Motion Object-Centric planning (TMOC), a grounded TAMP algorithm that learns to ground objects and their physical properties with a physics engine [17].

Recently, I have been focusing on improving robotic reasoning and planning capabilities with foundation models. In LLM+P, we investigated how to use LLMs to translate a complicated planning problem (in natural language) into PDDL so as a classical planner can be used to solve it [18]. COWP is an open-world task planning framework that uses knowledge from LLMs for robot situation handling [19]. We further extend the usage of LLMs to the motion level, where we introduced LLM-GROP, an algorithm extracting commonsense knowledge from LLMs for object rearrangement using task and motion planning [20]. To benchmark high-level perception and reasoning with foundation models, we proposed OpenEQA, the first open-vocabulary benchmark dataset for Embodied Question Answering supporting both episodic memory and active exploration use cases [21].

2 Background

Within this section, key concepts related to this dissertation are examined. The initial focus is on automated planning (classical AI planning), which will be used as the main tool for robot high-level task planning. Subsequently, there is an introduction of continuous motion planning from the robotics perspective, where we will cover the main concept of motion planning as well as two widely used sampling-based motion planning algorithms (i.e., RRT and PRM). Finally, we discuss the robot platform being utilized in this research.

2.1 Classical Planning

Task planning, or AI planning, is a important field in artificial intelligence. It involves generating a sequence of actions that leads to a specific goal while satisfying certain constraints. Essentially, the planning process uses a formal representation of the world state, along with actions and their potential outcomes, to find a path from an initial state to a goal state. As a process, it provides service robots with the capability to create a sequence of actions, allowing them to operate autonomously in dynamic and open-world environments.

Planning Domain Definition Language (PDDL): The Planning Domain Definition Language (PDDL), initiated by Drew McDermott and his associates in the late 90s, was created to streamline research in automated planning [2]. PDDL aims to standardize the formula-

tion of planning problems, thereby enhancing the comparability of different planning systems and algorithms by ensuring they are tackling identical issues.

PDDL structures planning problems around states and actions. States are depicted by logical propositions, while actions are determined by their preconditions (the circumstances necessary for their execution) and their effects (the changes in the world state resulting from the action). A PDDL planning problem encompasses an initial state, a group of actions, and a goal state. The primary task of a PDDL planner is to identify a series of actions that metamorphose the initial state into the goal state.

PDDL expresses planning problems in terms of states and actions. States are defined by logical propositions, and actions are defined by their preconditions (what must be true for the action to be executed) and their effects (what changes in the world state as a result of the action). A planning problem in PDDL consists of an initial state, a set of actions, and a goal state. The goal of a PDDL planner is to find a sequence of actions that transforms the initial state into the goal state.

In practical use, a PDDL problem will have two separate files: a domain file and a problem file. The domain file contains the definition of all the actions, including their preconditions and effects. The problem file describes the initial state and the goal state.

For a service robot, using PDDL means that it can reason about actions it should take based on its understanding of the world's state and its goal. It represents an effective and flexible tool for defining and solving planning problems in open world scenarios.

Answer Set Programming (ASP): Answer Set Programming (ASP) is a declarative programming paradigm rooted in research on non-monotonic logic and logic programming [22,

23]. Initially designed for problem-solving using the concept of "answer sets" or stable models, ASP is particularly effective for tackling combinatorial search problems.

The power of ASP lies in its ability to frame problems in terms of logical rules. An ASP solver then seeks solutions that comply with all given rules. These solutions, termed "answer sets," epitomize models of the program, encompassing a set of facts validated by the prescribed rules.

In ASP, problems are encapsulated by a logic program composed of a set of rules. Each rule features a head and a body. The notion is that the rule's head is valid if all the conditions in the rule's body are met. Utilizing this principle, ASP solvers compute the answer sets.

Emerging from research on non-monotonic logic and logic programming, ASP provides a robust language for defining problems. The logic rules encode the problems, and the ASP solver computes their solutions, i.e., the "answer sets." An answer set is essentially a model of the problem's logic program and thus signifies a solution.

ASP finds its utility in automated planning, particularly when defining complex planning problems using a high-level, logic-based language. This characteristic makes it especially valuable for service robots operating in environments that require intricate reasoning or knowledge handling with inherent uncertainty. Through ASP, these robots can formulate sets of actions (plans) that satisfy multiple constraints and achieve the desired goals.

Comparison of PDDL and ASP: PDDL and ASP, both effective in representing and resolving planning issues, rely on a declarative, logic-based methodology. This allows for a flexible problem representation and the application of advanced reasoning techniques.

The significant divergence resides in their expressive capacity and solving strategies. PDDL, as a language, accentuates the representation of states and their transitions induced by actions. This makes it ideal for typical AI planning problems, characterized by an initial state, a goal state, and an intervening set of actions.

ASP, conversely, provides for more intricate problem definitions due to its rule-based structure. It is equipped to tackle issues that demand sophisticated reasoning techniques, such as managing defaults and exceptions. The solving procedure in ASP revolves around identifying models that adhere to a specific set of rules rather than merely discovering a sequence of actions that lead to a goal.

The practical choice between PDDL and ASP is predicated upon the unique requirements of the problem being examined. PDDL is potentially more apt for problems that can be interpreted naturally as state transitions, while ASP might be a better fit for problems that call for complex logical reasoning.

From my experience, contemporary PDDL solvers are inclined to deliver a single solution. This approach, while being computationally efficient, inherently limits the exploration of all possible solutions. A more exhaustive study of the solution space could reveal superior or diverse plans that satisfy the defined constraints.

In contrast, ASP offers heightened flexibility. With configurations that can present all possible plans, ASP solvers provide a thorough set of solutions. This comprehensive view enables a rigorous evaluation and comparison of various strategies, potentially leading to optimal or diverse solutions that address the needs.

While not asserting the dominance of one over the other, this comparison clarifies a marked operational distinction between PDDL and ASP. The choice between the two

should reflect the specific demands of the problem. A PDDL solver might be more suitable if the focus is on computational efficiency and finding a single feasible solution. Conversely, if the problem calls for an extensive examination of all possible solutions for a detailed evaluation, an approach centered on ASP may be more appropriate.

2.2 Motion Planning

Motion planning is a fundamental concept in robotics concerned with finding a sequence of valid movements that would get an object from the start point to the goal point without colliding with any obstacles in the environment. This aspect of robotics is especially critical in service robotics, where robots must navigate complex, dynamic, and open worlds while performing tasks.

The challenge of motion planning increases exponentially with the dimensionality of the problem, hence it often relies on algorithms and mathematical models. Two popular techniques used in motion planning are the Rapidly-exploring Random Trees (RRT) [24] and the Probabilistic Roadmaps (PRM) [25].

RRT: Rapidly-exploring Random Trees (RRT) is an efficient, randomized data structure designed for a broad class of path planning problems. The core idea of RRT is that it builds a tree rooted at the starting configuration by using random samples from the search space. As a "greedy" algorithm, it has a preference for exploring unvisited regions.

An RRT grows a tree in the configuration space by starting at the robot's initial configuration (root of the tree) and expanding towards randomly chosen configurations. The tree eventually spans much of the reachable space, and with a high likelihood, it will reach a

point near the target configuration. Once this happens, the path from root to target configuration provides a solution to the motion planning problem.

PRM: Probabilistic Roadmaps (PRM) is another popular method for motion planning. Unlike RRT, which builds a tree, PRM constructs a graph (the roadmap) over the entire planning space. It works by taking random samples from the configuration space of the robot, testing them for validity (e.g., collision), and using a local planner to attempt to connect these configurations to other nearby configurations.

PRM tends to work well in multiple query scenarios, where there is a significant preprocessing phase, after which a series of goal configurations are given. The roadmap constructed can then be used to find paths quickly for any given start and goal configuration pair provided they are connected in the roadmap.

Both RRT and PRM are examples of sampling-based planning methods, which solve high-dimensional problems by avoiding an explicit representation of the configuration space. Instead, they generate samples and then connect these samples in a manner that tries to capture the connectivity of the free space.

2.3 Mobile Manipulation Platform

Our research mainly uses a mobile manipulator robot platform. It is a Segway-based mobile robot platform, the RMP110, for experimental trials. This mobile manipulator is equipped with a UR5e robot arm and a Robotiq Hand-E gripper. The platform incorporates a SICK laser sensor for tasks such as mapping, localization, and navigation. Additionally, an Astra Orbbec RGB-D camera is employed for human detection, interaction, and a wrist

camera is used for object detection and vision-based object grasping.

The robot's software operates on the Robot Operating System (ROS) [26] and is built upon the publicly available Building Wide Intelligence codebase [27]. This allows the robot to navigate and manipulate objects, including grasping and ungrasping, using MoveIt [28] and GGCNN.

Additionally, we have developed a simulation platform based on the Gazebo physics engine [29]. A difference is the use of the robotiq 2F-85 gripper in the simulation, as the urdf model for the Hand-E is unavailable.

3 Multimodal Embodied Attribute Learning

3.1 Introduction

Intelligent robots are able to interact with objects through exploratory actions in real-world environments. For instance, a robot can use a *look* action to figure out if an object is RED using computer vision methods. However, vision is not sufficient to answer if an opaque bottle is FULL or not, and actions that support other sensory modalities, such as *lift* and *shake*, become necessary. Given the sensing capabilities of robots and the perceivable properties of objects, it is important to develop algorithms to enable robots to use multi-modal exploratory actions to identify object properties, answering questions such as "*Is this object* RED *and* EMPTY?" In this article, we use **attribute** to refer to a perceivable property of an object and use **behavior**[1] to refer to an *exploratory action* that a robot can take to interact with the object. [2]

Given multimodal perception capabilities, a robot still needs to decide which of its many exploratory behaviors to perform on an object. In other words, the robot needs to generate an action policy for each given language request, as illustrated in Figure 3.1. For instance, to obtain an object's color, a robot could adjust the pose of its camera, whereas

[1]The terms of "behavior" and "action" are widely used in developmental robotics and sequential decision-making communities respectively. In this article, the two terms are used interchangeably.

[2]Project webpage: https://sites.google.com/view/attribute-learning-robotics/

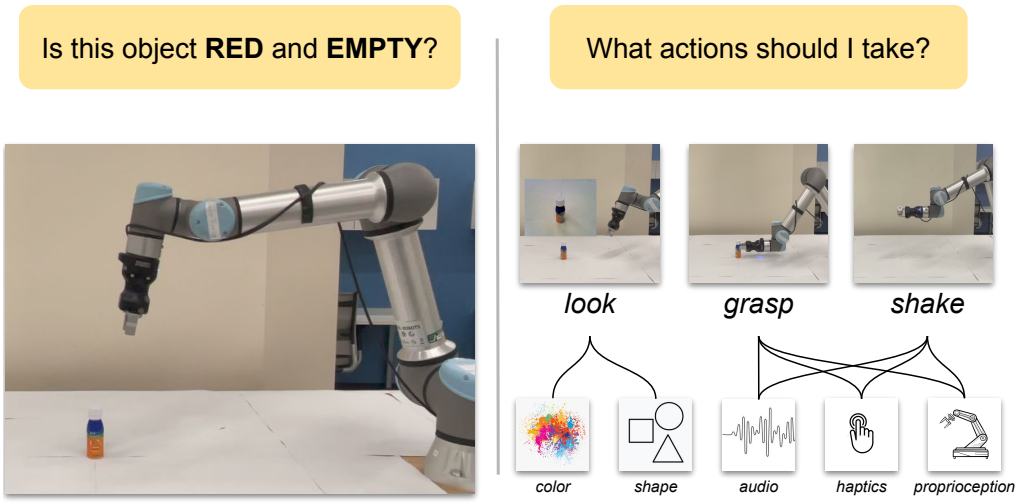


Figure 3.1: A robot is tasked with identifying if an object is RED and EMPTY. Given the various sensory modalities produced by exploratory behaviors, the robot must decide what behavior(s) to perform to gain maximum information.

sensing the content of an opaque container requires two behaviors: *grasp* and *shake*. The robot has to select actions in such a way that the information gain about object attributes is maximized while the cost of behaviors is minimized. Sequential reasoning is required in this action selection process. For example, *shake* would make sense only if *grasp* has been successfully executed. Also, robot perception capabilities are imperfect, so the robot sometimes needs to take the same behavior more than once. The above-mentioned observations motivate the development of this article focusing on the **M**ultimodal **E**mbodied **A**ttribute **L**earning (MEAL) problem. We informally define MEAL as follows:

> *Algorithms for* MEAL *problems aim to learn a policy for sequentially selecting exploratory behaviors to efficiently and accurately identify perceivable attributes of objects. Those behaviors might involve multiple sensory modalities and are not necessarily always successful.*

The capability of solving MEAL problems is important for robot multimodal perception. In this article, we introduce and investigate two types of MEAL problems: offline and online. Algorithms for **OFFLINE-MEAL** problems aim to learn robot behavioral exploration policies from a previously collected dataset. Probabilistic planning algorithms aim at computing action policies to help select actions toward maximizing long-term utility such as information gain in our case, while considering the uncertainty resulting from non-deterministic action outcomes. Markov decision processes (MDPs) [30] and partially observable MDPs (POMDPs) [31] enable an agent to plan under uncertainty with full and partial observability respectively. However, the observability of real-world domains is frequently mixed: some components of the current state can be fully observable while others are not. A mixed observability Markov decision process (MOMDP) is a special form of POMDP that accounts for both fully and partially observable components of the state [32]. In the **ONLINE-MEAL** setting, the robot needs to learn an accurate and efficient action policy for interacting with objects without pre-collected data. As a result, algorithms for **ONLINE-MEAL** problems are required to work on attribute classification and identification at the same time. ONLINE-MEAL raises the fundamental trade-off between exploration and exploitation. The robot has a short-term goal of identifying object attributes in the current task, and a long-term goal of improving its identification accuracy over multiple tasks. An extreme solution is to let the robot optimize its actions focusing on only the short-term goal. In doing so, the robot still improves its performance in identification tasks over a long term as the robot collects data along the way. However, the learning process in this extreme solution can be poor with respect to regret minimization, because it lacks a mechanism for actively improving its long-term attribute identification performance.

- This article introduces **M**ixed **O**bservability **R**obot **C**ontrol (MORC), as the **first algorithmic contribution** of this article, where we model OFFLINE-MEAL problems using MOMDPs because of the mixed observability of the world that the robot interacts with. For example, whether an object is in the robot's hand or not is fully observable, but object attributes such as color and weight are not.

- The **second algorithmic contribution** of this article is an algorithm which is called **MORC** with **I**nformation-**T**heoretic **R**eward **S**haping (MORC-ITRS) for ONLINE-MEAL problems. MORC-ITRS, for the first time, equips a robot with the capability of optimizing its sequential action selection toward efficiently and accurately classifying and identifying attributes at the same time.

MORC and MORC-ITRS are evaluated using three datasets: **ISPY32** [33] which contains 32 objects with eight exploratory behaviors and six types of sensory modalities; **ROC36** [34] which includes 36 objects with eleven behaviors and four types of modalities; **CY101** [35] which has 101 objects with ten behaviors and seven types of modalities. These datasets have previously been used for a variety of tasks including language grounding [36, 37], object recognition [38], object categorization [39], and sensorimotor learning [40]. Experiments on OFFLINE-MEAL problems show that the policies from MORC improve accuracy for recognizing new objects' attributes while reducing exploration cost, in comparison to baseline strategies that deterministically or randomly use predefined sequences of behaviors. For ONLINE-MEAL settings, compared with existing methods from the MEAL literature which also include a variant of MORC, MORC-ITRS reduces the overall cost of exploration in the long term while reaching a higher accuracy of attribute identification.

Initial versions of the MORC and MORC-ITRS algorithms were introduced in two separate conference papers [7, 41]. Both papers aimed to enable a robot manipulator to identify object attributes using multiple exploratory behaviors and the produced multimodal sensory data. Both papers modeled the non-determinism of action outcomes, and exploration costs using a sequential decision-making framework under partial observability. Despite the shared goals, the two conference papers were developed under different assumptions. Next, we describe the relationship between this article and the two previous papers in the three dimensions of "problem," "algorithm," and "evaluation."

- **Problems**. This article aims to solve two problems called OFFLINE-MEAL and ONLINE-MEAL. The two problems were initially defined under the respective names of Robot Attribute Identification (RAI) and Online Robot Attribute Learning (On-RAL) [41]. We observed that the initial problem statements [41] were incomplete, because neither the objective function nor the reward function was included in the definitions. We have fixed the issues in this article, and renamed the problems for better clarity.

- **Algorithms**. This article develops two algorithms called MORC and MORC-ITRS. Our naming strategy highlights that MORC-ITRS is based on MORC. The key idea of MORC was initially presented in a conference paper [7], where it was only informally described. MORC-ITRS was initially presented in another [41] under the name of ITRS. This article formally defines both algorithms using the terminology specified in the corresponding problem statements.

- **Evaluations**. The evaluation of MORC [7] was based on the two datasets of **ISPY32** and **ROC36**. The evaluation of MORC-ITRS [41] was based on the two datasets of

ISPY32 and **CY101**. The different datasets in the two conference papers made it hard to directly compare the different methods. In this article, new experiments were performed to include all three datasets in the evaluations.

In addition, we discuss related work more comprehensively and point out the limitations of both algorithms while identifying research directions for future work.

The remainder of this article is organized as follows. Section 3.2 discusses existing research on MEAL-related topics. Section 5.3 formally defines three MEAL problems, including the OFFLINE-MEAL and the ONLINE-MEAL. Section 3.4 presents existing research on **A**ction-**C**onditioned **A**ttribute **C**lassification (ACAC), which serves as a building block of algorithms and systems developed in this article. Sections 3.5 and 3.6 describe two algorithms respectively: MORC for OFFLINE-MEAL and MORC-ITRS for ONLINE-MEAL. Experimental results and demonstrations are detailed in Section 3.7. The article is concluded in Section 3.8, including discussions about the limitations of our algorithms and directions for future work.

3.2 Related Work

This section summarizes a few research areas that are relevant to Multimodal Embodied Attribute Learning (MEAL), the focus of this article. We first briefly describe the concept of "attribute", which is widely used in the computer vision community, and then discuss existing research on robot perception (unimodal and multimodal). After that, a representative sample of algorithms for planning under uncertainty is described, based on which we develop MORC and MORC-ITRS. Finally, we present existing work on object-centric robot

perception, which is the application domain of this article.

Visual Attribute Learning The word "attribute" refers to as "an inherent characteristic" of an object in the computer vision community [42]. Attributes are also semantic and machine-understandable properties that are used by people to describe images [43]. Early vision-based attribute learning methods used image segments to learn visual attributes as patterns [44]. Later, researchers studied visual attribute learning in the context of generalization across object categories [45], and considered transferring visual attributes for previously unseen object classes [46]. More recent approaches focused on zero-shot or few-shot learning for specifying attributes [47, 48, 49], relative attributes to enable visual comparisons between images [50, 43], and attributes for an unconstrained set of objects by providing large-scale datasets [51, 52, 53]. In contrast to traditional attribute learning using these vision methods, we focus on Multimodal Embodied Attribute Learning (MEAL), where objects are explored and attributes are detected by a physical robot. Modern robots have the capability to actively interact with objects, producing rich sensory signals that go beyond vision.

Early research on symbol grounding introduced the term of "category" as an invariant feature of objects and events from their sensory projections [6]. In this article, we focus on object-centric robot perception, and use "attribute" to refer to the categorical representations of object features that are perceivable through a robot's multimodal exploratory behaviors.

Unimodal Perception in Robotics Among various modalities that have been researched in robotics, visual perception has been widely studied, including visual manipulation [54] and navigation [55]. Language is another important modality that robotics researchers frequently focus on, solving the challenge of grounding natural language into noisy percepts and physical actions (as reviewed in a recent article [56]). Tactile sensing is traditionally studied in the area of sensors, but recent papers have investigated more about correlations with robot actions, as reviewed in articles [57, 58]. There are also a few works that demonstrate the usefulness of smell for robots, especially in the application domain of gas source localization [59]. A limited number of papers have even mimicked the sense of taste for the interaction and cognitive abilities of modern robots [60]. Every single modality has been shown its effectiveness towards improving the perception capability of intelligent robots. Our agent learns from sensory modalities such as vision, audio, and haptics, and works on the problem of robot multimodal perception, which is reviewed in detail in the next paragraph.

Multimodal Perception in Robotics Significant advances have been achieved recently in computer vision [61, 62] and natural language processing [63, 64]. While language and vision are important communication channels for robotic perception, many object attributes cannot be detected using vision alone [65] and people are not always available to verbally provide guidance in exploration tasks. Therefore, researchers have jointly modeled language and visual information for multimodal text-vision tasks [66]. However, many of the most common nouns and adjectives such as SOFT and EMPTY have a strong non-visual component [67] and thus, robots need to perceive objects using additional sensory modal-

ities to reason about and perceive such linguistic descriptors. To address this problem, several lines of research have shown that incorporating a variety of sensory modalities is the key to further enhancing the robotic capabilities in recognizing multisensory object attributes, as reviewed in recent articles [68, 58]. For example, visual and physical interaction data yields more accurate haptic classification for objects [69], and non-visual sensory modalities, such as audio and haptics, coupled with exploratory behaviors such as *touch* or *grasp*, have been shown useful for object recognition [70, 71, 72]. Grounding natural language descriptors that people use to refer to objects has also been a promising method for attribute recognition problems [33, 73]. More recently, researchers have developed end-to-end systems to enable robots to learn to perceive the environment and perform actions at the same time [74, 75]. A major limitation of these and other existing methods is that they require large amounts of object exploration data, which is much more expensive to collect as compared to vision-only datasets. A few approaches have been proposed to actively select actions at test time, for instance, when recognizing an object [76, 39]. One recent work has also shown that robots can bias which behavior to perform at training time, that is, when learning a model grounded in multiple sensory modalities and behaviors, but they did not learn an actual policy for doing so [37]. Different from existing work, MORC for OFFLINE-MEAL problems is for learning an action policy when deciding whether a set of attributes hold true for an object. MORC-ITRS for ONLINE-MEAL problems learns an action policy for object exploration that a robot can use when learning to ground the semantics of attributes.

Planning under Uncertainty Decision-theoretic methods have been developed to help agents plan actions and address uncertainty in non-deterministic action outcomes [30, 77]. Existing planning models such as partially observable Markov decision process (POMDP) [31], belief space planning [78] and Bayesian approaches [79] have shown great advantages for planning robot perception behaviors, because robots need to use exploratory behaviors to estimate the current world state. To learn semantic attributes, robots frequently need to choose multiple actions, so POMDP which is useful for long-term planning is particularly suitable. Many of the POMDP-based robot perception methods are vision-based [80, 81, 82, 83]. Compared to those methods, our robot takes advantage of non-visual sensory modalities, such as audio and haptics. Our proposed algorithms both modeled the mixed observability [32] in domains of a robot interacting with objects. The main difference between OFFLINE- and ONLINE-MEAL is that the former assumed that sufficient training data and annotations are available for the robot to learn the perception models of its exploratory behaviors, and the latter deals with data collection and task completion processes simultaneously.

Object-centric Robot Perception To select actions to identify objects' perceivable attributes such as HEAVY, RED, FULL, and SHINY, robots need observation models for their exploratory behaviors. Researchers have developed algorithms to help robots determine the presence of possibly new attributes [84] and learn observation models of objects' perceivable attributes given different exploratory behaviors [33, 36, 85]. In the case where the object attributes refer to the object's function, they are then referred to as 0-order affordances [86]. Those methods focused on learning to improve the robots' perception

capabilities. Once the learning process is complete, a robot can use the learned attributes to perform tasks, such as attribute identification, for example, to tell if a bottle is HEAVY and RED. Compared to those learning methods, we consider both OFFLINE- and ONLINE-MEAL, where the robot learns the attribute classifiers (an exploration process) and uses the learned classifiers to identify object attributes (an exploitation process). Because ONLINE-MEAL agents iteratively learn and identify attributes, the exploration-exploitation trade-off is a fundamental decision-making challenge in this unknown robot perception environment. While the problem has been studied in multi-armed bandit [87] and reinforcement learning settings [77], it has not been studied in MEAL contexts.

3.3 Problem Definitions

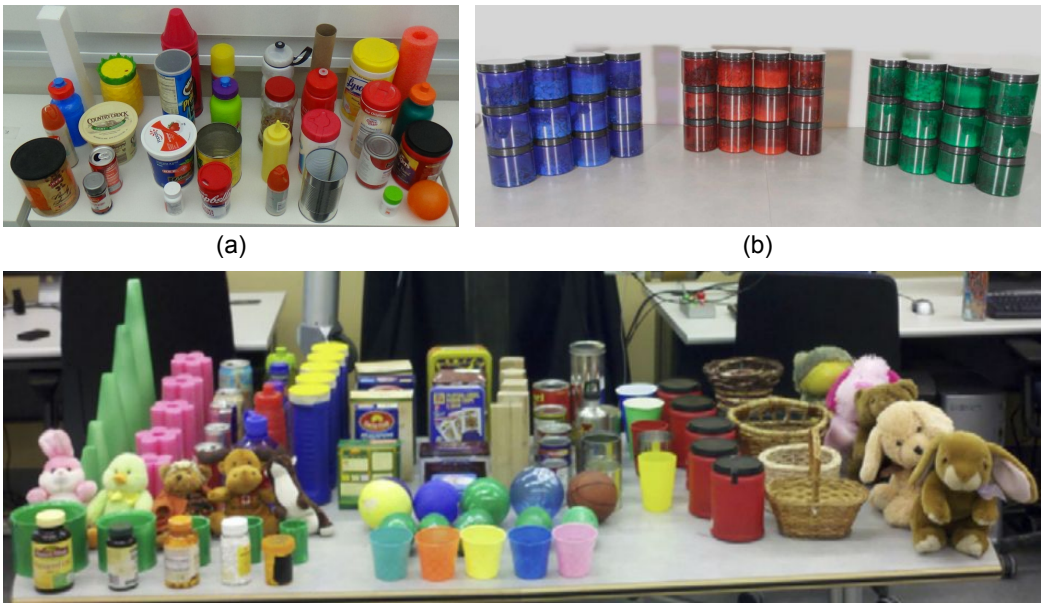
In this section, we first introduce the terminology of our work. Then we formally define three types of robot multimodal perception problems.

A robot has a set of behaviors, such as *look*, *push*, and *lift*, that can be used for interacting with everyday objects as shown in Figure 3.2. Let $o \in Obj$ be an object and $a \in \mathscr{A}^e$ be an *exploratory behavior*. Examples of a robot executing some of the exploration behaviors are shown in Figure 3.3.

Each exploratory behavior is coupled with a set of sensory modalities, e.g., vision, haptics, and audio. We use $m \in \mathscr{M}$ to refer to a sensory modality. This behavior-modality

Table 3.1: Table of Notation.

Symbol/Notation	Definition
o	An object
Obj	The set of all objects
a	A behavior
$\mathscr{A}^e$	The set of all exploratory behaviors
m	A modality
$\mathscr{M}$	The set of all modalities
$\Gamma(a)$	A behavior and modality coupling function
$\mathscr{M}_a$	A set of modalities that a behavior a produces
f_a^m	A sensory data instance of a modality m being recorded when a behavior a is applied
f_a	A set of sensory data instances from all modalities ($m \in \mathscr{M}_a$) that a robot receives after performing a
N^m	Dimensionality for a modality m
P	The set of all attributes
p	An attribute
v^p	The Boolean value of an attribute p indicating if p applies to an object
$ID(p,o)$	An attribute identification function
$\mathbf{v}$	Values of a set of attributes
$\mathbf{p}$	A set of attributes
$\mathscr{A}^r$	The set of all reporting actions
$\mathscr{A}$	The set of all actions including exploratory behaviors and reporting actions
$\mathscr{S}$	The global state space
$\mathscr{X}$	The state set specified by fully observable domain variables
x	A fully observable state
$\mathscr{Y}$	The state set specified by partially observable domain variables
y	A partially observable state
N^p	The number of queried attributes
$\mathscr{Z}$	A set of observations including $\emptyset$ (none) observation
$\mathscr{Z}^h$	A set of observations excluding $\emptyset$ (none) observation
$R(s,a)$	The reward function
t_a	Time length for executing a behavior a
r^-	Negative reward by given an incorrect report action
r^+	Positive reward by given a correct report action
ξ	An episode for representing a single attribute identification task
$s \oslash a$	An operation that outputs true (or false) when the identification task is successful (or not)
$Rst(\xi)$	A function that outputs if a task is successful
$Cst(\xi)$	A function that outputs the accumulative action cost in a task
$\mathscr{D}$	A dataset where each instance is in the form of $(f_a, p) : v^p$
$\Psi(f_a, p)$	A binary classifier. A robot collect data instance f_a on an object after performing action a, and $\Psi(f_a, p)$ outputs if attribute p applies to this object
$\hat{C}$	An action cost budget
$\Phi(f_a^m, p)$	A binary classifier that is similar to $\Psi(f_a, p)$ but with modality specifications
α	A normalization constant
w_a^m	A weight for a binary classifier $\Phi(f_a^m, p)$
Θ_p^a	A confusion matrix that are computed by cross-validation at training stage
$T_{\mathscr{X}}$	Transition functions for fully observable states in MOMDP
$T_{\mathscr{Y}}$	Transition functions for partially observable states in MOMDP
γ	A discount factor in MOMDP
$O(s,a,z)$	An observation function that specifies the probability of observing z when action a is executed in state s
$v_s^{p_i}$	The true value of the i^{th} attribute in state s
$v_z^{p_i}$	The observed value of the i^{th} attribute in observation z
$Ent(s,a)$	The entropy that is used for indicating the perception quality



(a) (b) (c)

Figure 3.2: Everyday objects in the three datasets that are used in the experiments of this article: (a)ISPY32 [33] (b)ROC36 [34] (c)CY101 [35].

coupling is formulated using function Γ:

$$\mathscr{M}_a = \Gamma(a) \tag{3.1}$$

where $\mathscr{M}_a \subseteq \mathscr{M}$. For example, $\{audio, haptics, vision\} = \Gamma(push)$ means that behavior *push* produces signals from audio, haptics, and vision modalities. When a is performed on an object o, for each $m \in \mathscr{M}_a$, the robot is able to record a data instance $f_a^m \in \mathbb{R}^{N^m}$, where N^m is the dimensionality for the modality m. Table 3.2 shows the set of viable combinations of behavior-modality pairs for one of the datasets used in our experiments (detailed in Section 3.7), along with the feature dimensionality N^m. We use f_a to represent a set of sensory data instances from all modalities ($m \in \mathscr{M}_a$) that a robot receives after performing a.

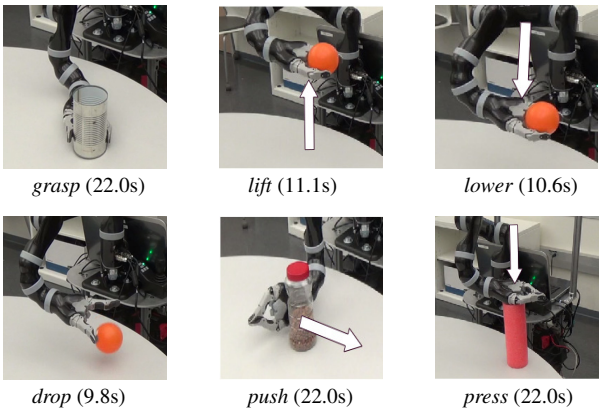


Figure 3.3: Examples of behaviors and their durations in seconds (behaviors are from the ISPY32 dataset detailed in Section 3.7).

Let $\mathscr{P}$ specify a set of attributes that are used to describe objects in a domain. Given an object o, v^p is either *true* or *false*, depending on if p applies to o, where we use $ID(p,o)$ to refer to this attribute identification function. Here we "override" the function ID to use it to process a set of attributes:

$$\mathbf{v} = ID(\mathbf{p}, o) \tag{3.2}$$

where $\mathbf{v} = [v^{p_0}, v^{p_1}, \cdots]$, $\mathbf{p} = [p_0, p_1, \cdots]$, and v^{p_i} is the value of the i^{th} attribute of object o. For instance, given a red empty object (i.e., o) and two attributes $[\text{BLUE}, \text{EMPTY}]$ (i.e., $\mathbf{p}$), $ID([\text{BLUE}, \text{EMPTY}], a\ red\ empty\ object)$ outputs $[false, true]$. The reporting action set $\mathscr{A}^r$ includes actions that are used for the robot to report $\mathbf{v}$ to the human user. For the same red empty object, there will be two binary variables specifying whether each of the attributes is true or false. As a result, there will be four reporting actions corresponding to the four combinations of the attributes' values. The action set $\mathscr{A} = \mathscr{A}^e \cup \mathscr{A}^r$.

The robot state space is mixed observable and has two components, $\mathscr{X}$ and $\mathscr{Y}$. The

Table 3.2: The number of features extracted from each combination of robot behavior and perceptual modality for ISPY32. "VGG" modality is computed from 2D image of the object and is deep visual features from the 16-layer VGG network [33].

Behavior	Modality		
	color	shape	VGG
look	64	308	4096
	audio	haptics	proprioception
grasp	100	60	20
drop, hold, lift, lower, press, push	100	60	

global state space S includes a Cartesian product of $\mathscr{X}$ and $\mathscr{Y}$,

$$S = \{(x,y) \mid x \in \mathscr{X} \text{ and } y \in \mathscr{Y}\} \tag{3.3}$$

$\mathscr{X}$ is the state set specified by fully observable domain variables. In our case, $x \in \mathscr{X}$ represents the current state of the robot-object system, e.g., whether *lift* and *drop* behaviors are successful or not, or whether the object is in hand or not (i.e., the effect of behavior *grasp*). $\mathscr{Y}$ is the state set specified by partially observable domain variables. In our case, these variables correspond to the values of object attributes that are queried about, i.e., $\mathbf{v}$. Thus, $\mid \mathscr{Y} \mid = 2^{N^p}$, where N^p is the number of queried attributes, i.e., $\mid \mathbf{p} \mid$. For instance, given an object description that includes three attributes (e.g., $[\text{RED}, \text{EMPTY}, \text{BOTTLE}]$), $\mathscr{Y}$ includes $2^3 = 8$ states. Since $y \in \mathscr{Y}$ is partially observable, it needs to be estimated through observations, which will be defined in the next paragraph. Note that there is no state transition in the space of $\mathscr{Y}$, as we assume object attributes do not change over executions of robot actions.

Let $\mathscr{Z} = \mathscr{Z}^h \cup \emptyset$ be a set of observations. Elements in $\mathscr{Z}^h$ include all possible combinations of object attributes (i.e., $\mathbf{v}$) and have one-to-one correspondence with elements in

$\mathscr{A}^r$. For instance, when $\mathbf{p} = [\text{RED}, \text{EMPTY}, \text{BOTTLE}]$, there exists an observation $z \in \mathscr{Z}^h$ that is $[true, false, true]$ that represents "the object's color is red; it is not empty, and it is a bottle." Behaviors that produce no information gain (including those failed behaviors) and reporting actions in $\mathscr{A}^r$ result in a $\emptyset$ (none) observation.

$R \to \mathbb{R}$ is the reward function. Each *exploration behavior*, $a^e \in \mathscr{A}^e$, has a cost that is determined by the time required to complete the behavior. These costs are empirically assigned according to the datasets used in this research. The costs of *reporting actions* depend on whether the report is correct.

$$R(s,a) = \begin{cases} -t_a, \text{if } s \in S,\, a \in \mathscr{A}^e \\ r^-, \text{if } s \in S,\, a \in \mathscr{A}^r,\, s \oslash a = false \\ r^+, \text{if } s \in S,\, a \in \mathscr{A}^r,\, s \oslash a = true \end{cases} \tag{3.4}$$

where t_a is the time length for executing the behavior a, r^- (or r^+) is negative (or positive) given an incorrect (or correct) report. $s \oslash a$ outputs true (or false) when the identification task is successful (or not). We further define an episode as $\xi = [s_0, a_0, r_0, s_1, a_1, r_1, \cdots]$ for representing a single attribute identification task, where $s_i \in S$, $a_i \in \mathscr{A}$, and $r_i \in R$. Function $Rst(\xi)$ outputs if a task ξ is successful:

$$Rst(\xi) = \begin{cases} 1, \text{if there exists } s_i, a_i \in \xi,\, s_i \in S,\, a_i \in \mathscr{A}^r,\, s_i \oslash a_i = true \\ 0, \text{otherwise} \end{cases} \tag{3.5}$$

Function $Cst(\xi)$ outputs the accumulative action cost in task ξ:

$$Cst(\xi) = \sum_{a_i \in \mathscr{A}^e} t_{a_i} = - \sum_{a_i \in \mathscr{A}^e} r_i(s, a_i) \tag{3.6}$$

Definition 1 (**ACAC**). *At training time, the input includes a set of labeled sensory data instances, each in the form of* $(f_a, p) : v^p$. *This training set is sufficiently large and denoted as* $\mathscr{D}$. *Solving an* ***Action-Conditioned Attribute Classification***[3] *(ACAC) problem produces a binary classifier:*

$$\Psi(f_a, p), \text{ for each pair of } a \in \mathscr{A}^e \text{ and } p \in \mathscr{P}$$

At testing time, given an object o, a robot collects data instances f_a *after performing action a and* $\Psi(f_a, p)$ *outputs* true *or* false *estimating if attribute p applies to o.*

Definition 2 (**OFFLINE-MEAL**). *Solving an* **OFFLINE-MEAL** *problem produces a policy* π *that sequentially selects action* $a \in \mathscr{A}$ *to identify the value of:*

$$ID(\boldsymbol{p}, o), \text{ given } \mathscr{D}, R(s, a)$$

where the objective is to maximize the number of successful task completions within a cost budget $\hat{C}$. *Let* $[\xi_0, \xi_1, \cdots, \xi_n]$ *be a set of n attribute identification tasks, then the objective*

[3] We use attribute classification to refer to the problem of learning the attribute classifiers, which is a supervised machine learning problem. We use attribute identification to refer to the task of identifying whether an object has a set of attributes or not, which corresponds to a sequential decision-making problem.

function can be defined as follows:

$$\arg\max_{\pi}(E(\sum_{i\leq n} Rst(\xi_i))), \text{where } \sum_{i\leq n} Cst(\xi_i) < \hat{C} \tag{3.7}$$

Definition 3 (ONLINE-MEAL). *Solving an* **ONLINE-MEAL** *problem produces a policy* π *that sequentially selects action* $a \in \mathscr{A}$ *to identify the value of:*

$$ID(\boldsymbol{p}, o), \text{ given } R(s,a)$$

OFFLINE- *and* ONLINE-MEAL *share the same objective function (Equation 3.7), while algorithms for* ONLINE-MEAL *are* not *provided with pre-collected data* $\mathscr{D}$*. At execution time, after performing a to identify* $\boldsymbol{p}$*, the robot collects data* f_a*. After each identification task, the robot receives* $\boldsymbol{v}$*, the values of attributes* $\boldsymbol{p}$*.*

Remark 1: MEAL agents are developed under the following assumptions (detailed in this section and highlighted here):

- the state space is mixed observable: the fully observable component is predefined and the partially observable component is specified by the queried attributes.
- robot actions and their realizations are predefined, including knowledge about the target object' precise location on the table.
- there is uncertainty in action outcomes, simulated by a certain probability.
- the goal is to maximize both long-term utility (lifelong learning) and short-term utility (maximizing success rate of the current task).

Remark 2: Rational OFFLINE-MEAL agents treat individual attribute identification tasks independently, whereas rational ONLINE-MEAL agents learn from the data collected in early tasks, trading off early-phase performance for long-term performance.

Remark 3: One might have noticed the difference between the objective function defined in Equation 3.7 and the reward function in Equation 3.4. There can be the question of whether our reward function supports the robot in achieving the objective or not. Note that the objective function presented in Equation 3.7 includes two dimensions: identification accuracy and exploration cost. The reward function presented in Equation 3.4 includes three terms, where action execution time t_a corresponds to the exploration cost, and rewards r^- and r^+ correspond to the identification accuracy. Altogether, this reward function is able to motivate the robot to maximize identification accuracy and minimize exploration costs at the same time. Thus, our current reward function supports computing policies towards achieving the objective.

3.4 Preliminaries

The algorithms developed in this article rely on existing research on Action-Conditioned Attribute Classification (ACAC). For the sake of completeness, we summarize the technical approach of previous ACAC work [34, 33], which is used as a building block for defining algorithms for MEAL problems.

We define **individual classifiers** for connecting data instances f_a^m to each attribute p, denoted as $\Phi(f_a^m, p)$. We assume that the outputs of Φ can be mapped to probabilities, i.e., $\Phi(f_a^m, p)$ can estimate $\mathbf{Pr}_a^m(v^p = true \mid f_a^m)$. In line with prior research [34, 33], individual classifiers Φ were structured using support-vector machines with a polynomial kernel in

this article. The binary classifier Ψ (introduced in Section 5.3) is for connecting a set of $\Phi(f_a^m, p)$ regardless of modality specifications. Similarly, the outputs of Ψ can be mapped to probabilities, i.e., $\Psi(f_a, p)$ can estimate $\mathbf{Pr}_a(v^p = true \mid f_a)$. It should be noted that for each behavior, different modalities are not equally preferred when the robot identifies certain attributes. For instance, color is more useful than shape when identifying RED. Thus, the probability estimates of Φ are combined using weighted combination and normalized again to compute the final probability estimates of Ψ:

$$\mathbf{Pr}_a(v^p = true \mid f_a) = \alpha \times \sum w_a^m \times \mathbf{Pr}_a^m(v^p = true \mid f_a^m)$$

where α is a normalization constant to ensure the probabilities sum up to 1.0 and $w_a^m \in [0.0, 1.0]$ is a reliability weight indicating how good the classifier associates with the behavior a and the modality m is at recognizing attribute p. In other words, each behavior acts as a classifier ensemble where each individual classifier's output is combined using a weighted combination. The weights are estimated by performing cross-validation of the classifier specific to that modality and behavior.

At the end of the training stage, cross-validation at the behavior level is used to compute the confusion matrix $\Theta_p^a \in \mathbb{R}^{2\times 2}$ for each pair of attribute p and behavior a. These confusion matrices are normalized to compute the True Positive, True Negative, False Positive, and False Negative rates for each behavior-attribute pair. Cross-validation is not a general step for ACAC, but will later be used in MORC. Example confusion matrices are shown in Figure 3.4. Next, we describe our MORC approach to address the problem of OFFLINE-MEAL which is defined in Section 5.3.

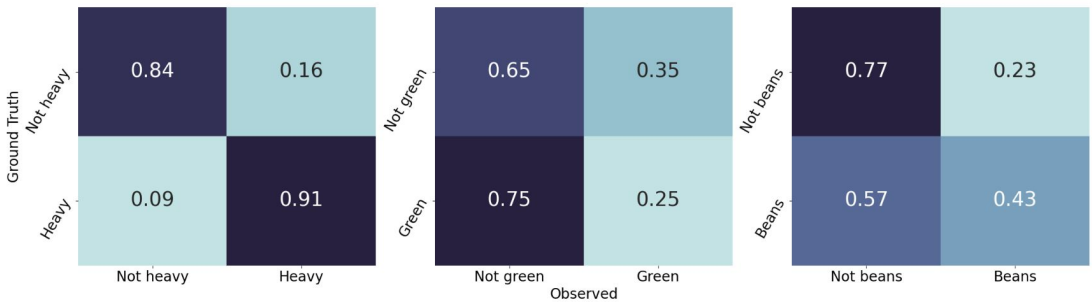


Figure 3.4: Example confusion matrices training from 36 objects (ROC36) showing the TP, FP, TN, and FN rates for three of the attributes when using the robot's *shake* behavior. The behavior is good at recognizing HEAVY due to the rich haptic feedback produced when shaking an object, somewhat good at recognizing BEANS (referring to the objects' contents) due to the sound produced by the contents, and poor at recognizing GREEN as no visual input is processed when performing this behavior.

3.5 An Algorithm for OFFLINE-MEAL: MORC

In this section, we describe the theoretical framework of mixed observability robot control (MORC) for solving OFFLINE-MEAL problems. Behaviors such as *look* and *drop*, have different costs and different accuracies in attribute recognition. At each step, the robot has to decide whether more exploration behaviors are needed, and, if so, select the exploration behavior that produces the most information. In order to sequence these behaviors toward maximizing information gain, subject to the cost of each behavior (e.g., the time it takes to execute it), it is necessary to further consider preconditions and non-deterministic outcomes of the behaviors. For instance, *shake* and *drop* behaviors make sense only if a preceding unreliable *grasp* behavior succeeds.

In this article, we assume action outcomes are fully observable and object attributes are not. For instance, a robot can reliably sense if a *grasp* behavior is successful, but it cannot reliably sense the color of a bottle or if that bottle is FULL. Due to this mixed observability

and unreliable action outcomes, we use mixed observability MDPs (MOMDPs) [32] to model the sequential decision-making problem for object exploration.

A MOMDP has mixed state variables. The fully observable state components are represented as a single state variable x (in our case, the *robot-object status*, e.g., the object is in hand or not), while the partially observable components are represented as state variable y (in our case, the *object attributes*, e.g., the object is HEAVY or not). As a result, (x, y) specifies the complete system state, and the state space is factored as $S = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the space for fully observable variables and $\mathcal{Y}$ is the space for partially observable variables.

Formally, a MOMDP model is specified as a tuple,

$$(\mathcal{X},\ \mathcal{Y},\ \mathcal{A},\ T_{\mathcal{X}},\ T_{\mathcal{Y}},\ R,\ \mathcal{Z},\ O,\ \gamma),$$

where $\mathcal{A}$ is the action set, $T_{\mathcal{X}}$ and $T_{\mathcal{Y}}$ are the transition functions for fully and partially observable variables respectively, R is the reward function, $\mathcal{Z}$ is the observation set, O is the observation function, and γ is the discount factor that specifies the planning horizon.

3.5.1 Action Transition System

$T_{\mathcal{X}} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the state transition function in the fully observable component of the current state. $T_{\mathcal{X}}$ includes a set of conditional probabilities of transitions from $x \in \mathcal{X}$—the fully observable component of the current state—to $x' \in \mathcal{X}$, the component of the next state, given $a \in \mathcal{A}$ the current action. The transition diagram is shown in Figure 3.5. Reporting actions and illegal exploration behaviors (e.g., *drop* an object in

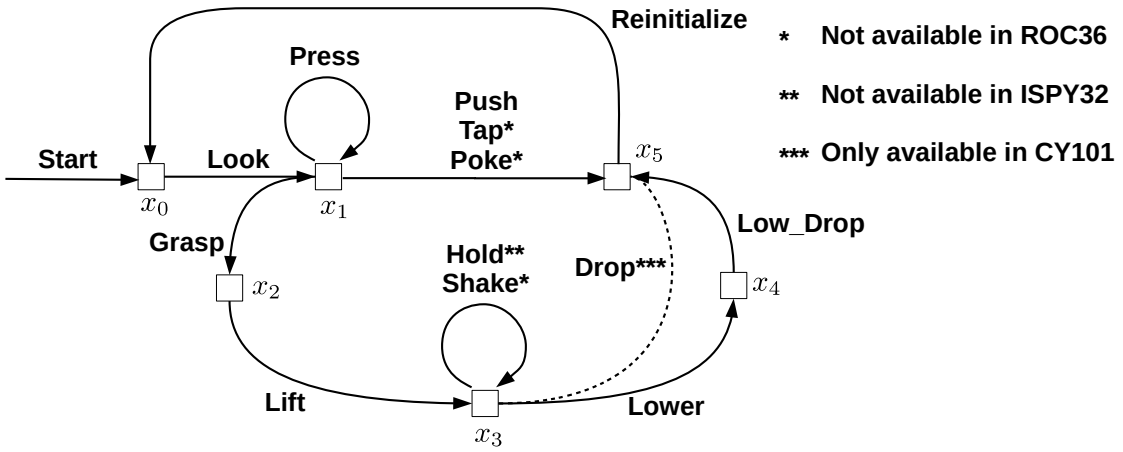


Figure 3.5: A simplified version of the transition diagram in space $\mathscr{X}$ for object exploration. This figure only shows the probabilistic transitions led by *exploration behaviors*. *Report actions* that deterministically lead transitions from $x_i \in \mathscr{X}$ to *term* (terminal state) are not included.

state x_1—before a successful *grasp*) lead state transitions to *term* (terminal state) with 1.0 probability.

Most exploration behaviors are unreliable and succeed probabilistically. For instance, $T_{\mathscr{X}}(x_4, drop,\ x_5) = 0.95$ in our case, indicating there is a small probability the object is stuck in the robot's hand (detailed in Section 3.7.1). Such non-deterministic action outcomes are considered in our experiments. The success rate of the behavior *look* is 1.0 in our case since without changing positions of either the camera or the object it does not make sense to keep running the same vision algorithms.

$T_{\mathscr{Y}} : \mathscr{Y} \times \mathscr{A} \times \mathscr{Y} \to [0,1]$ is the state transition function in the partially observable component of the current state. It is an identity matrix in our case, (we assume) because object attributes do not change during the process of the robot's exploration behaviors.

3.5.2 Observation Function

$O : S \times \mathscr{A} \times \mathscr{Z} \rightarrow [0,1]$ is the observation function that specifies the probability of observing $z \in \mathscr{Z}$ when action a is executed in state s: $O(s,a,z)$. In this article, the probabilities are learned from performing cross-validation on the robot's training data. As described in Section 3.4, ACAC produces confusion matrix classifiers $\Theta_p^a \in \mathbb{R}^{2\times 2}$ for each attribute p and each action a.

$$O(s,a,z) = \mathbf{Pr}(\mathbf{v}_z \mid \mathbf{v}_s, a) = \prod_{i=0}^{N^p-1} \Theta_{p_i}^a(v_s^{p_i}, v_z^{p_i}) \tag{3.8}$$

where $\Theta_p^a \in R^{2\times 2}$ is a confusion matrix for attribute p and action a; $\mathbf{v}_s$ and $\mathbf{v}_z$ are the true and observed values of the attributes; $v_s^{p_i}$ (or $v_z^{p_i}$) is the true (or observed) value of the i^{th} attribute; and N^p is the total number of attributes in the query. The robot might fail in exploratory actions. In that case, the robot receives an empty observation, which causes no belief change.

So far, we have specified all components of MORC. Next, we discuss a way of computing high-quality policies for OFFLINE-MEAL that include large numbers of attributes.

3.5.3 Dynamically-Learned Controllers

The OFFLINE-MEAL problem can include a prohibitively large number of attributes. One of the datasets in our experiments contains 81 attributes, resulting in 2^{81} possible states in $\mathscr{Y}$. It is computationally intractable to generate a far-sighted policy while considering all the attributes. A strategy called iCORPP was introduced for dynamically constructing minimal (PO)MDPs to model domain attributes for robot planning [88]. Behind iCORPP is

Algorithm 1 MORC

Require: $\mathscr{P}$; $T_{\mathscr{X}}$; $\mathscr{A}^e$; Sol; $R(s,a)$; $\mathscr{D}$

Take queried attribute(s) $\mathbf{p}$ from human, where $\mathbf{p} \subseteq \mathscr{P}$
Generate $\mathscr{X}$, $\mathscr{Y}$, $\mathscr{A}^r$, $T_{\mathscr{Y}}$ and $\mathscr{Z}$ using $\mathbf{p}$
Compute confusion matrix Θ_p^a using $\mathscr{D}$ where $p \in \mathbf{p}$, $a \in \mathscr{A}$
Generate $O(s,a,z)$ with Θ_p^a for $p \in \mathbf{p}$ using Eqn. 3.8
Compute policy π using Sol for ($\mathscr{X}$, $\mathscr{Y}$, $\mathscr{A}$, $T_{\mathscr{X}}$, $T_{\mathscr{Y}}$, R, $\mathscr{Z}$, O, γ)
Uniformly initialize belief b
while Current state s is not $term$ **do**
    Select action a with π based on b, and execute a
    Make an observation z where $z \in \mathscr{Z}$
    Update b with z and a using Bayesian rule
end while

a family of algorithms for Integrated commonsense Reasoning and probabilistic Planning (IRP) [89]. Those algorithms aim to decomposing a sequential decision-making problem into two tractable subproblems that focus on high-dimensional reasoning (e.g., objects with many attributes) and long-horizon planning (e.g., tasks that require many actions). In line with iCORPP, we model only those attributes necessary to the current query in MORC, where the goal is to include a relatively small set of attributes in our MOMDPs while maintaining the quality of the generated policies.

Algorithm 1 shows the complete process of MORC[4]. We dynamically construct MOMDP controllers by specifying the following components in order: 1) State set $\mathscr{Y}$ that includes only the attributes that are mentioned in the query (e.g., BLUE, HEAVY, and BOTTLE, given that a user asks whether an object is *a blue heavy bottle* or not); 2) State set S, the Cartesian product of $\mathscr{X}$ (predefined) and $\mathscr{Y}$; 3) Action set $\mathscr{A}^r$, where each reporting action $a^r \in \mathscr{A}^r$ corresponds to a state in $\mathscr{Y}$; 4) Action set $\mathscr{A}$, union of $\mathscr{A}^e$ (predefined) and $\mathscr{A}^r$; 5) $\mathscr{Z}^h$, object attribute combinations; 6) $\mathscr{Z}$, union of $\mathscr{Z}^h$ and $\emptyset$. The components together form a complete MOMDP that is relatively very small, and typically includes fewer than 100

[4]Source code: https://github.com/keke-220/Predicate_Learning

states at runtime. Our approach enables automatic generation of complete MOMDP models, which can be encoded, as in our experiments, such that existing planning algorithms (e.g., [90]) can be used to generate policies.

The implementation of the transition system of MORC is introduced in Section 3.5.1. Figure 3.5 also presents how we manually specify transition diagrams for different datasets. The observation function (detailed in Section 3.5.2) depends on classifiers produced by ACAC. When learning those classifiers, MORC assumes that sufficient training data and annotations are available for the robot. However, a large amount of object exploration data is very expensive for robots to collect in the real world. Our solution is to interleave ACAC and attribute identification where the robot interacts with the current object, collects exploration data, and uses the data to improve attribute classifiers for future identifications. This online process is referred to as ONLINE-MEAL, which will be discussed in the next section.

3.6 An Algorithm for ONLINE-MEAL: MORC-ITRS

In this section, we describe MORC with information-theoretic reward shaping (MORC-ITRS), which is a novel algorithm that is built on MORC but focuses on balancing exploration and exploitation in ONLINE-MEAL problems. An overview of MORC-ITRS as applied to ONLINE-MEAL is presented in Figure 3.6.

3.6.1 Information-Theoretic Reward

We first introduce the shaped information-theoretic reward function in MORC-ITRS. In ONLINE-MEAL problems, the robot needs to optimize its actions toward not only im-

proving the accuracy of attribute identification but also minimizing the cost of exploratory behaviors. We introduce the two factors of *perception quality* and *interaction experience* into the reward design of MOMDPs to achieve the trade-off between exploration (actively collecting data for attribute learning) and exploitation (using the learned attributes for identification tasks).

Let $Ent(s,a)$ be the entropy of the distribution over $\mathscr{Z}$, given s and a, which is used for indicating the **perception quality** of exploratory behavior a over the y component of s:

$$Ent(s,a) = -\sum_{z_i \in \mathscr{Z}} O(s,a,z_i) \log_2 O(s,a,z_i) \tag{3.9}$$

where z_i is the i^{th} observation and $O(s,a,z_i)$ is the probability of observing z_i in state s after taking action a. $O(s,a,z_i)$ is computed using the data instances gathered in the ONLINE-MEAL process.

Let $IE(p,a)$ be the **interaction experience** of applying action a to identify attribute p, which is in the form of:

$$IE(p,a) = \frac{1}{\delta} \cdot \mid \{f \in f_a, labeled(f,p) = true\} \mid \tag{3.10}$$

where f_a is a set of instances that a robot has collected so far, and $labeled(f,p)$ returns *true* if f has been labeled w.r.t. p, where the value v^p is *true* or *false*. δ is a sufficiently large integer to ensure $IE(p,a)$ is in the range of [0,1). A lower value of $IE(p,a)$ reflects a higher need of further exploring (p,a).

Building on the concepts of perception quality and interaction experience, our information-

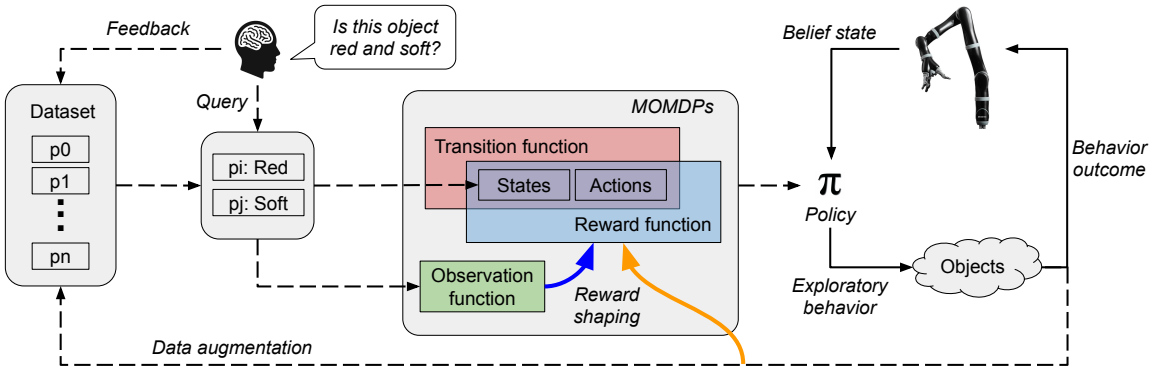


Figure 3.6: An overview of the MORC-ITRS algorithm. A human user will choose an object and ask a query such as "*Is this object* RED *and* SOFT*?*". The robot will generate a perception model on the specified attributes, i.e., RED and SOFT. Queried attributes and the corresponding perception model then will be used to construct states and the observation function of the MOMDP model respectively. The reward function will be shaped by the quality of the observation function and the robot's experience. The robot uses the generated MOMDP model to compute a policy π and interacts with the queried object. Newly-perceived feature data will be used to update the robot's experience and augment the dataset. Humans will give feedback to the robot's answer and attach labels to the feature data points.

theoretic reward function is defined as follows:

$$R^{IT}(s,a) = R(s,a) + \alpha \cdot Ent(s,a) - \beta \cdot IE(p,a) \tag{3.11}$$

where α and β are natural numbers and used for adjusting how much perception quality and interaction experience are considered in reward shaping. Informally, when $O(s,a,z)$ is close to being uniform, the perception model of (s,a) is poor, and the value of $Ent(s,a)$ is high. As a result, our new reward function will encourage the robot to take action a by offering extra reward $\alpha \cdot Ent(s,a)$. When the robot is experienced in applying a to identify attribute p, $IE(p,a)$ will be high. In this case, an extra penalty of $\beta \cdot IE(p,a)$ will discourage the robot from taking those well-explored behaviors. In comparison to standard MOMDPs, where reward and observation functions are independently developed, MORC-ITRS enables

Algorithm 2 MORC-ITRS

Require: $\mathscr{P}$; $T_{\mathscr{X}}$; $\mathscr{A}$; Sol; α; β; $R(s,a)$

1: Initialize $IE(p,a) = 0$ for each action $a \in \mathscr{A}$ and $p \in \mathscr{P}$
2: Initialize online training dataset $\mathscr{D} = \emptyset$
3: **repeat**
4: Take queried attribute(s) $\mathbf{p}$ from human, where $\mathbf{p} \subseteq \mathscr{P}$
5: Generate $\mathscr{X}$, $\mathscr{Y}$, $T_{\mathscr{Y}}$, and $\mathscr{Z}$ using $\mathbf{p}$
6: Compute confusion matrix Θ_p^a using $\mathscr{D}$ where $p \in \mathbf{p}$, $a \in \mathscr{A}$
7: Generate $O(s,a,z)$ with Θ_p^a for $p \in \mathbf{p}$ using Eqn. 3.8
8: Compute $Ent(s,a)$ using Eqn. 3.9
9: Generate $R^{IT}(s,a)$ with $R(s,a)$ using Eqn. 3.11
10: Compute policy π using Sol for $(\mathscr{X},\ \mathscr{Y},\ \mathscr{A},\ T_{\mathscr{X}},\ T_{\mathscr{Y}},\ R^{IT},\ \mathscr{Z},\ O,\ \gamma)$
11: Initialize action set $\mathscr{A}^{select}$ and feature set $\mathscr{F}$ with $\emptyset$
12: Uniformly initialize belief b
13: **while** Current state s is not $term$ **do**
14: Select action a with π based on b, append a to $\mathscr{A}^{select}$, and execute a
15: Record data instances f_a, and $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_a\}$
16: Make an observation z where $z \in \mathscr{Z}$
17: Update b with z and a using Bayesian rule
18: **end while**
19: Ask human to provide $\mathbf{v}$ for $\mathbf{p}$ as label(s) for $\mathscr{F}$
20: **for** each a in $\mathscr{A}^{select}$ **do**
21: Update $\mathscr{D}$ using $\mathscr{F}$ and $\mathbf{v}$
22: Update $IE(p,a)$ for $a \in \mathscr{A}$ and $p \in \mathbf{p}$ using Eqn. 3.10
23: **end for**
24: **until** end of interactions

the reward function to adapt to the changes of the observation function.

3.6.2 Algorithm Description

Algorithm 2 presents MORC-ITRS for active ONLINE-MEAL problems. The inputs of MORC-ITRS include attribute set $\mathscr{P}$, transition function $T_{\mathscr{X}}$, action set $\mathscr{A}^e$, MOMDP solver Sol, parameters α and β, naive reward function $R(s,a)$. MORC-ITRS does not have a termination condition.

MORC-ITRS starts with initializing the interaction experience function with zeros for all (p,a) pairs, and then initializes dataset $\mathscr{D}$ that will be later augmented as the robot interacts with objects (Lines 1 and 2). In each iteration of the main loop (Lines 3-24), MORC-ITRS

takes an identification query from people (Line 4), constructs a MOMDP model (Lines 5-10), computes its policy, uses the policy to interact with an object (Lines 13-18), and augments its dataset for improving the MOMDP model in the next iteration (Lines 20-23).

In the first inner loop (Lines 13-18), the robot interacts with an object based on the generated policy. π suggests an action at each state b. The robot then executes the action and makes an observation. Based on the action and observation, the robot updates its belief using the Bayesian rule. After selecting each action, MORC-ITRS records this action along with its collected feature data (Lines 14 and 15). In Line 19, we ask people to provide the label y for the collected data. The final step is to iterate over all selected actions to augment $\mathscr{D}$, and calculate the new interaction experience (Lines 20-23).

Intuitively, we aim to encourage the robot to select exploratory behavior $a \in \mathscr{A}^e$ from those behaviors, where the perception model of (s,a) is of poor quality, and there is relatively limited experience of applying a to attribute p, i.e., the experience of (p,a) is limited.

3.7 Experiments

In this section, we present the experiment setup and experimental results from the evaluation of our MORC and MORC-ITRS algorithms. MORC assumes the availability of training data for learning action-conditioned attribute classifiers. Accordingly, the baselines for evaluating MORC includes:

- *Random*: Actions are randomly selected from both reporting and legal exploration actions. A trial is terminated by any of the reporting actions.
- *Random Legal*: Actions are randomly selected from legal exploration actions. Under

an exploration budget, one selects the reporting action corresponding to y with the highest belief. This baseline corresponds to the algorithm for MEAL problems of [37] (we did not use their linguistic component).

- *Predefined*: An action sequence is strictly followed: *ask*, *look*, *press*, *grasp*, *lift*, *lower*, and *drop*. Under an exploration budget or in early terminations caused by illegal actions, the robot selects the reporting action that makes the best sense.
- *Predefined Plus*: The same as Predefined except that unsuccessful actions are repeated until achieving the desired result(s).

MORC-ITRS assumes no prior data, so the robot must learn perception models and perform attribute identification at the same time. We compare MORC-ITRS with the baselines of:

- *Iterative Random Legal*: It is an iterative version of one of the baselines for solving OFFLINE-MEAL problems. Just like Random Legal, the robot considers only the "legal" behaviors (e.g., *lift* is legal only after a successful *grasp* behavior), and then randomly selects one from the legal actions. The only difference is that the robot collects more data after identifications, and uses the data for future tasks. With an exploration budget for each trial (50 seconds and 80 seconds for one-attribute trials, i.e., $N^p = 1$, and two-attribute trials, i.e., $N^p = 2$, respectively), the robot is forced to report $y \in \mathscr{Y}$ of the highest belief.
- *Iterative* MORC: It iteratively runs MORC using all data collected so far. This process is repeated after each batch. Iterative MORC enables ONLINE-MEAL actions by *passively* collecting data and training attribute classifiers.

3.7.1 Experiment Setup

Dataset Description: Three public datasets of **ISPY32** [33], **ROC36** [34], and **CY101** [35] are used in our experiments. CY101 (an updated version of the dataset [39]) contains many more household objects and attributes.

In the ISPY32 dataset, a robot from the Building-Wide Intelligence project [27] explored 32 objects using 8 exploratory behaviors: *look*, *grasp*, *lift*, *hold*, *lower*, *drop*, *push*, and *press* (Figure 3.3). The *hold* behavior was performed by holding the object in place. The *look* behavior was performed by taking a visual snapshot of the object using the robot's sensors prior to exploration. Each behavior was performed 5 times on each object in the dataset. Features of VGG, color, SURF, auditory, finger, and haptics were recorded in ISPY32.

In ROC36, the robot explored 36 different objects using 11 prototypical exploratory behaviors: *look*, *grasp*, *lift*, *shake*, *shake-fast*, *lower*, *drop*, *push*, *poke*, *tap*, and *press* 10 different times per object. The objects are lidded containers with the same shape and varied along 3 different attributes: 1) color: RED, GREEN, BLUE; 2) weight: LIGHT, MEDIUM, HEAVY; and 3) contents: BEANS, RICE, GLASS, SCREWS. These variations result in the $3 \times 3 \times 4 = 36$ objects bearing combinations of these attributes in the set $\mathscr{P}$ that the robot is tasked with learning.

For CY101 dataset, an uppertorso humanoid robot with 7-DOF arm explored 101 objects belonging to 20 different categories using 10 exploratory behaviors: *look*, *grasp*, *lift*, *hold*, *shake*, *drop*, *push*, *tap*, *poke*, and *press*. Seven different types of features including auditory, vibrotactile, finger, color, optical flow, SURF, and haptics (i.e., joint forces) were

considered in CY101. Each behavior was performed 5 times per object.

Every individual classifier (introduced in Section 3.4) corresponds to an attribute-behavior pair. The exact numbers of the classifiers needed by the robot depend on the datasets that provide different numbers of attributes. For instance, in experiments using the **ROC36** dataset, there were a total of $9 \times 11 = 99$ classifiers.

Action Costs and Action Failures: Each exploratory behavior a has a cost (planning and execution) in the range of [0.5, 22.0] that came with the datasets, and is modeled in $R(s,a)$. For instance, the cost of behavior *press* (22.0) is much higher than the cost of behavior *look* (0.5). The costs of behaviors in the three datasets are different because the datasets were collected using different robots. Additionally, action *ask* has the cost of 100.0.[5]

In MORC, the reward of the reporting action was +500.0 (or -500.0) when the robot's attribute identification is correct (or incorrect). In MORC-ITRS, we set the reward to be +300.0 (or -300.0). Most actions are considered unreliable to some degree in our MOMDP model and we uniformly set the failure probability to 0.05 which we did not refine in OFFLINE- or ONLINE-MEAL. For instance, an unsuccessful *drop* behavior models the situation that the object is stuck in the robot's hand. We used an off-the-shelf system for solving MOMDPs [90]. γ is a discount factor and $\gamma = 0.99$ in our case. This setting gives the robot an unspecified, relatively long planning horizon.

We observed different behaviors of the robot given different success bonuses and failure penalties. Intuitively, increasing the success bonus (positive reward) and the failure penalty (negative reward) encourages the robot to spend more time exploring objects for higher

[5]Action *ask* was used only in the ISPY32 experiments, because other exploration behaviors are not as effective as in ROC36 and CY101.

success rates, i.e., being risk-averse. We also observed that a very large success bonus frequently produced optimistic, risk-seeking behaviors, such as reporting before taking any exploration behaviors. Such discussions point to the research area of reward engineering, which is a long-standing challenge to researchers working on planning under uncertainty and reinforcement learning. In this article, the success bonus and failure penalty values are manually specified.

3.7.2 MORC Evaluation

Next, we describe the experiments we conducted to evaluate MORC. We aim to answer the following questions:

- *How does* MORC *perform in efficiency and accuracy*? If MORC is more efficient and more accurate than the baselines, then we can claim MORC's superiority in achieving the objective of solving OFFLINE-MEAL problems.
- *Can we build a "super" model in* MORC? A super model includes all potential attributes into the sequential decision maker. In comparison, MORC dynamically constructs query-dependent MOMDPs.

The training process of MORC follows the principle of "leave one object out." In other words, an object was randomly selected, and all data instances corresponding to the particular object were excluded in training the classifiers. The excluded object was then used for evaluating the classifiers' performance. We iterated over all the objects in the experiments for evaluating MORC in solving OFFLINE-MEAL problems. Specifically, for dataset **ISPY32** which includes 32 objects, each classifier was trained using $31 \times 5 = 155$ data

samples, where one object was excluded, and each behavior was repeated five times. For dataset **ROC36**, there were $35 \times 10 = 350$ data instances used for training each classifier.

Illustrative Trial of MORC

We now describe an example in which a robot works on an OFFLINE-MEAL task. We randomly selected an object from the ISPY32 dataset: a blue and red bottle full of water. We then randomly selected attributes, in this case YELLOW and METALLIC, and asked the robot to identify whether the object has each of the attributes or not. The selected object was not part of the robot's training set used to learn the attribute classifiers and the MOMDP observation model. The robot should report negative to both attributes while minimizing the overall cost of exploration behaviors.

Given this user query, the state space of MORC includes 25 states. We then generate an action policy using past work's methods [90]. Currently, building the model takes almost no time, and we uniformly gave five seconds for policy generation using the model (same in all experiments). The time for computing the policy is insignificant relative to the time for exploratory behaviors (which is what we are really trying to minimize).

Figure 3.7 shows the belief change in this process. The initial distributions over $\mathscr{X}$ and $\mathscr{Y}$ are $[1.0, 0.0, \cdots]$ and $[0.25, 0.25, 0.25, 0.25]$ respectively. The policy suggests to *look* first. We queried the dataset to make an observation, *neg-neg* in this case. The belief over $\mathscr{Y}$ is updated based on this observation: $[0.41, 0.28, 0.19, 0.13]$, where the entries represent *neg-neg*, *neg-pos*, *pos-neg*, and *pos-pos* respectively. There is a (fully observable) state transition in $\mathscr{X}$, from x_0 to x_1, so the belief over $\mathscr{X}$ becomes $[0.0, 1.0, 0.0, \cdots]$. Based on the updated beliefs, the policy suggests taking the *push* behavior, which results in another *neg-*

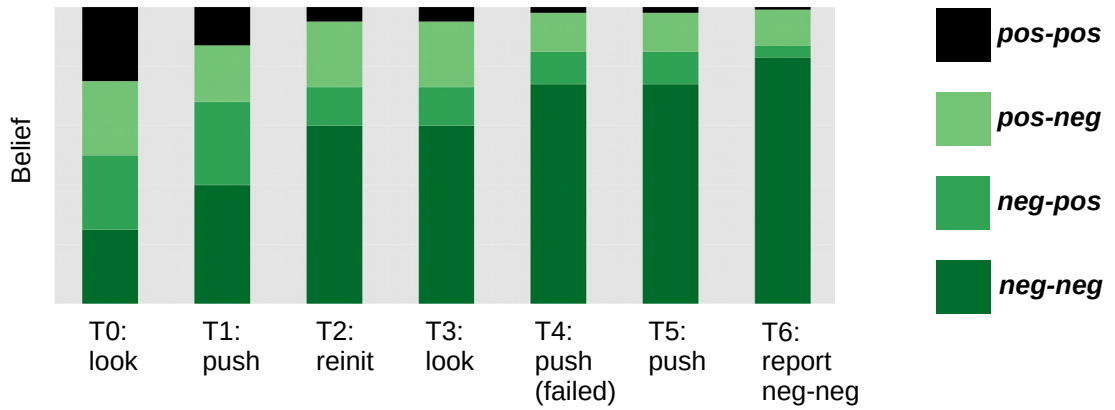


Figure 3.7: Action selection and belief change in the exploration of a red and blue bottle full of water using MORC, given a query of "is this object YELLOW and METALLIC?"

neg observation. Accordingly, the belief over $\mathscr{Y}$ is updated to $[0.60, 0.13, 0.22, 0.05]$, which indicates that the robot is more confident that the object is neither YELLOW nor METALLIC. After behaviors of *reinitialize*, *look*, *push*, and *push* (this first *push* behavior was unsuccessful, and produced the $\emptyset$ observation), the belief over $\mathscr{Y}$ becomes $[0.84, 0.04, 0.12, 0.01]$. The policy finally suggests reporting *neg-neg*, making it a successful trial with an overall cost of 167 seconds, which results in an overall reward of $500 - 167 = 333$ (an incorrect report would have resulted in -667 reward).

Remarks: It should be noted that the classifiers associated with each action and word will produce an output even in cases where the sensory signals from that action are irrelevant to the word. For instance, although the sensory signals relevant to *push* are haptics and audio, the first *push* behavior results in an observation of YELLOW. It was "YELLOW:neg", because most objects in the prior training set are not yellow. The robot favors behaviors that distinguish "easy" attributes (*look* distinguishes YELLOW well in this case). If a behavior is useful, the robot will prefer taking it early. The more the behavior is delayed, the more

the expected reward is discounted (we use a discount factor of 0.99 in our experiments).

Results of Applying MORC to OFFLINE-MEAL Problems

How does MORC perform in efficiency and accuracy on ROC36?

In each trial, we place an object that has three attributes (color, weight, and content) on a table and then generate an object description that includes the values of two or three attributes. This description matches the object in only half of the trials. When two (or three) attributes are queried, $\mathscr{Y}$ includes four (or eight) states plus the *term* state, resulting in $\mathscr{S}$ that includes 25 (or 49) states. The other components of MORC grow accordingly, given an increasing number of queried attributes.

Experimental results are reported in Table 3.3. Not surprisingly, randomly selecting actions produces low accuracy. The overall cost is smaller in more challenging trials (all three attributes are questioned), because in these trials there are relatively fewer exploratory behaviors (more attributes produce more reporting actions), making the agent more likely to take a reporting action. MORC reduces the overall action cost while significantly improving the reporting accuracy. Our performance improvement is achieved by repeating actions as needed, selecting legal actions (e.g., *lift* is legal only if the current state is x_2) that produce the most information or have the potential of doing so in the future, and even arbitrarily reporting without "wasting" exploratory behaviors given queries where the exploratory behaviors are not effective.

How does MORC perform in efficiency and accuracy on ISPY32? In this set of experiments, a user query is specified by randomly selecting one object and N^p attributes ($1 \leq N^p \leq 3$), on which the robot is questioned. Each data point is an average of 200 trials,

Table 3.3: Performances of MORC and two baseline planners in cost and accuracy on the ROC36 dataset. Numbers in parentheses denote the Standard Deviations over 400 trials.

N^p	Method	Overall cost (std)	Accuracy
2	Random	17.56 (30.00)	0.245
	Predefined Plus	37.10 (0.00)	0.583
	MORC (Ours)	29.85 (12.87)	0.860
3	Random	10.12 (21.77)	0.130
	Predefined Plus	37.10 (0.00)	0.373
	MORC (Ours)	33.87 (8.78)	0.903

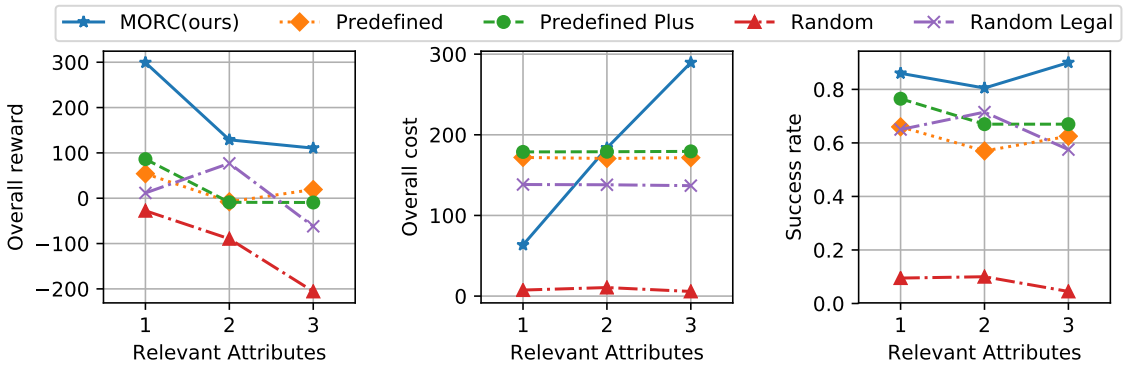


Figure 3.8: Evaluations of five action strategies (including MORC) on the ISPY32 dataset. Comparisons are made in three categories of *overall reward* (Left), *overall exploration cost* (Middle), and *success rate* (Right).

where we conducted pairwise comparisons over the five strategies, i.e., the strategies were evaluated using the same set of user queries. A trial is successful only if the robot reports correctly on all attributes. It should be noted that most of the contexts are misleading in this dataset due to the large number of object attributes, so more exploratory behaviors confuse the robot if the behaviors are not carefully selected.

Figure 3.8 shows the experimental results. The overall reward is computed by subtracting the overall action cost from the reward yielded by the reporting action (either a big bonus or a big penalty). We do not compute standard deviations in this dataset, because the diversity of the tasks results in problems of very different difficulties. We can see MORC

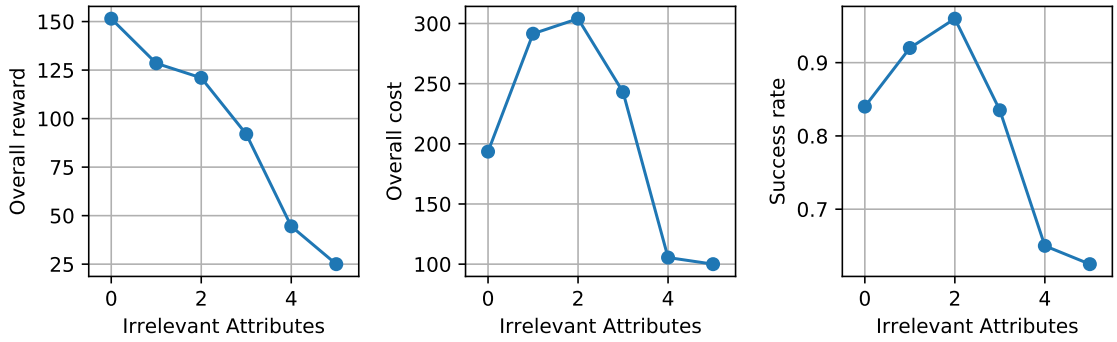


Figure 3.9: A "super" MORC framework that models two relevant attributes, and an increasing number of irrelevant attributes (x-axis). Our dynamically learned controllers correspond to the left end of each curve, and model only the relevant attributes. The three subfigures correspond to three different dimensions for evaluation: *overall reward* (Left), *overall exploration cost* (Middle), and *success rate* (Right).

consistently performs the best in terms of the overall reward and overall accuracy. When more attributes are queried, MORC enables the robot to take more exploratory behaviors (Middle subfigure), whereas the baselines could not adjust their question-asking strategy accordingly.

Can we build a "super" model in MORC? The last experiment aims to evaluate the need for dynamically constructed controllers, answering the question "*Can we build a 'super' controller that models all attributes?*" We constructed MOMDP controllers including two relevant and an increasing number of irrelevant attributes (i.e., the ones that are not queried). Our dynamically learned controllers include only the relevant attributes and correspond to the curves' left ends. Results are shown in Figure 3.9. We can see, the quality of the generated action policies decreases soon, e.g., from >150 to <25 in reward, when more irrelevant attributes are included in MORC. The right two subfigures show that MORC first tries to achieve higher accuracy by taking more exploration behaviors and then "gives

up" due to the growing number of irrelevant attributes. The results show the infeasibility of "super" controllers in MORC that model all attributes and justify the need for dynamic controllers.

3.7.3 MORC-ITRS Evaluation

Regarding the evaluation of MORC-ITRS for ONLINE-MEAL problems, we aim to answer the following questions:

- *How does* MORC-ITRS *perform in efficiency and accuracy?*
- *Does* MORC-ITRS *outperform baselines for individual attribute?*
- *How sensitive is* MORC-ITRS *to the parameters?*

Attributes: In order to select attributes that are learnable given the robot's exploratory behaviors, evaluations of all attributes in the two datasets were performed prior to the experiments. We set $|\mathscr{P}|$ to 10 and picked the attributes that have enough positive examples for training and those are most learnable.

Queries: At the beginning of each trial, N^p was either 1 or 2. At the end of each trial, the robot is told if the identification was correct. In the case of $N^p = 1$, the robot could learn the attribute's ground-truth value from the human's feedback. In the case of $N^p = 2$, the robot could do so, only if the 2D identification was correct.

Batch-based Learning: In both datasets, we randomly split the objects into three subsets of equal sizes. The subsets are used for pretraining (Obj^{pre}), training (Obj^{train}), and testing (Obj^{test}) respectively. In the pretraining phase, the robot started with a handcrafted policy where each action is forced to be applied on the queried object once. We collected

feature instances with labels from those interactions and built a pretraining dataset $\mathscr{D}^{pre}$ that represents the robot's prior knowledge.

In principle, we do not need a pretraining dataset in MORC-ITRS. In practice, however, without a small amount of data for "warm up," the robot might take a large number of interactions with objects for exploration in order to identify object attributes and learn meaningful observation models. This number is particularly large at the early learning phase. A practical challenge is that the datasets used in this article can provide only a limited number of samples for each attribute-behavior pair. As a result, we would have to reuse the same samples from the dataset when the robot performs the same actions more than N times ($N = 5$ in our case), which is detrimental to the quality of the experiments. To alleviate this practical issue, we provide a small amount of data for pretraining, though the ONLINE-MEAL algorithm does not require that.

Illustrative Trials of MORC-ITRS

Table 3.4: Early and late observation models for behavior *press*

	Early phase		Late phase	
	Not soft (Observed)	Soft (Observed)	Not soft (Observed)	Soft (Observed)
Not soft (Ground truth)	0.68	0.32	0.82	0.17
Soft (Ground truth)	0.50	0.50	0.20	0.80

From the robot's many trials of the learning experience, we selected two trials (T_1 and T_2), where the robot faced the same object (a Coke can that has attributes METAL, EMPTY, and CONTAINER) and needed to answer the same question "*Is this object* SOFT?" From the dataset, we know that the correct answer should be "no" (the robot did not know it). T_1 appeared at the second batch of training, and T_2 appeared at the ninth. We present both

trials and explain how the robot performed better in T_2.

In T_1 (early learning phase), the robot first performed the *look* behavior. Then, the robot had the following options: *grasp*, *tap*, *push*, *poke* and *press* according to Figure 3.5. Specifically, for *press*, the confusion matrix $\Theta^{press}_{\text{SOFT}}$ (shown in Table 3.4) was nearly uniform, which is typical in the early learning phase. Among those "less useful" behaviors, the robot chose *grasp*. The distribution over $\mathscr{Y}$ was changed from [0.37, 0.63] to [0.46, 0.54], where the entries represent "not soft" and "soft" respectively. After *press*, MORC-ITRS sequentially suggested *grasp*, *lift*, *hold* and *hold*. Finally, the robot reported *pos* that resulted in a failed trial with a total cost of 55.5 seconds.

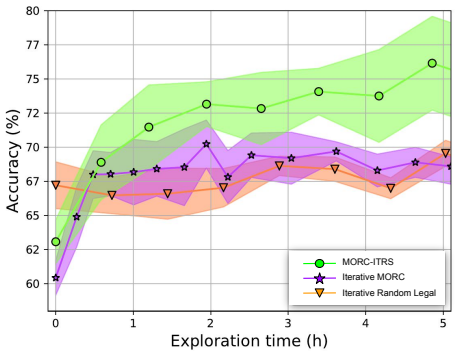
In T_2 (late learning phase), behavior *press* became more useful for identifying attribute SOFT compared to T_1, as shown in Table 3.4. For *grasp*, $IE(\text{SOFT}, grasp) = 0.67$ and $\Theta^{grasp}_{\text{SOFT}}$ was [0.66, 0,33, 0.61, 0.38] (TN, FN, FP, TP), which meant that the robot was experienced with behavior *grasp* and considered *grasp* was not as useful as *press*. Accordingly, MORC-ITRS suggested *press* instead of *grasp* after taking *look*. The belief over $\mathscr{Y}$ changed from [0.57, 0.43] to [0.67, 0.33]. After only *look* and *press*, the robot was able to quickly report *neg*, resulting in a successful trial with a total cost of 22.5 seconds.

From the above two trials (same query and object in different learning phases), we see how the improved perception model of $(press, \text{SOFT})$ helped the robot correctly identify SOFT with a low cost.

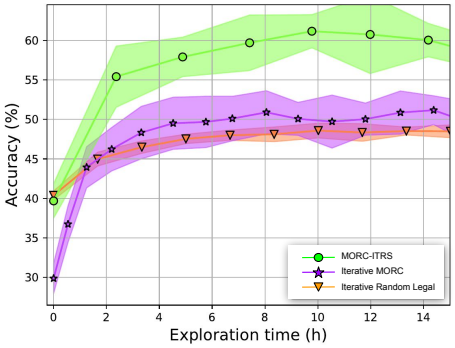
Results of Applying MORC-ITRS to ONLINE-MEAL Problems

How does MORC-ITRS perform in efficiency and accuracy? Figure 3.10 shows the learning curve for one-attribute and two-attribute identification queries evaluated on the

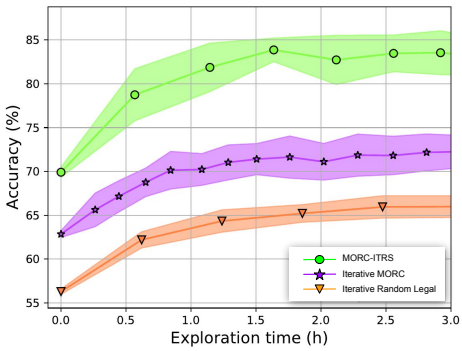
three datasets, where we conducted experiments over three different strategies (two baselines and MORC-ITRS). x-axis is the accumulative cost of all trials at the training phase. Since the cost is determined by the time required to complete each action, we can regard the x-axis as training time. y-axis reflects the identification accuracy at the testing phase. The proposed method consistently performs better in task completion rate along the whole training process and achieves higher accuracy than baselines.



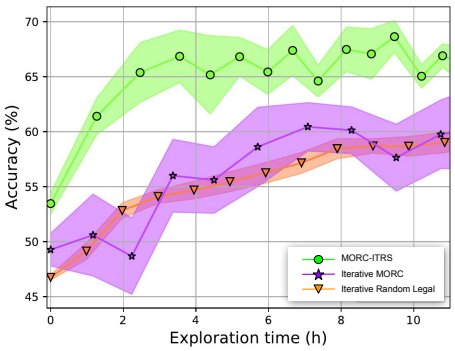
(a) One-attribute queries on **ISPY32**.



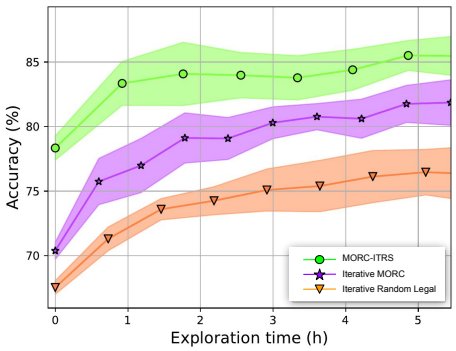
(b) Two-attribute queries on **ISPY32**.



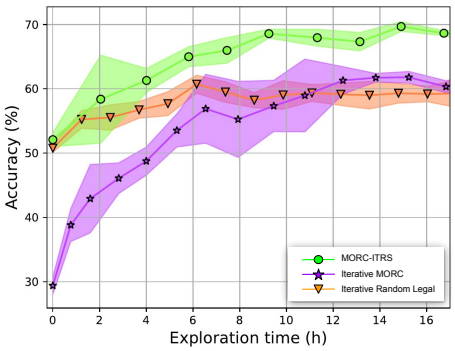
(c) One-attribute queries on **ROC36**.



(d) Two-attribute queries on **ROC36**.



(e) One-attribute queries on **CY101**.



(f) Two-attribute queries on **CY101**.

Figure 3.10: Time length of conducting exploratory actions in hours, and identification accuracy of ONLINE-MEAL tasks, where we compared MORC-ITRS (ours) to two baseline strategies including *Iterative Random Legal*, and *Iterative* MORC.

Although we provided the same pretraining data, three curves in the two subfigures (for each of the three datasets) do not start from the same point. That is because pretraining data only affects the observation model for the robot, but it is not directly related to the policy for attribute identification. Three strategies have the same observation model but they use different methods to select exploratory behaviors. As a result, the task-completion accuracy is not the same for them at the starting point. MORC-ITRS assigns extra rewards for exploration at the very beginning of the training phase. It resulted in not only a bigger cost but also a higher accuracy.

Does MORC-ITRS outperform baselines for individual attribute? At an exploration cost budget of 2 hours, we further evaluated the performance of each individual attribute on CY101 using the three strategies we mentioned, as shown in Figure 3.11, where 10 attributes are ranked by the identification accuracy of our method, i.e. MORC-ITRS. The robot has a higher identification accuracy for most of the attributes using MORC-ITRS, while the Iterative Random Legal baseline produces a relatively weak result compared to the other two strategies. Attributes such as PLASTIC, HARD, and EMPTY are more difficult to learn since the accuracy is no more than 80% for all three methods. And attributes such as BLUE, FULL and CONTAINER are easier, where Iterative MORC and MORC-ITRS both offer pretty good results.

How sensitive is MORC-ITRS to the α and β parameters? In Eqn. 3.11, we have two parameters α and β. We conducted experiments on CY101 with different α and β combinations, as shown in Figure 3.12. One observation is that the selections of α and β affect the performance of MORC-ITRS. A small α value leads to a higher identification accuracy

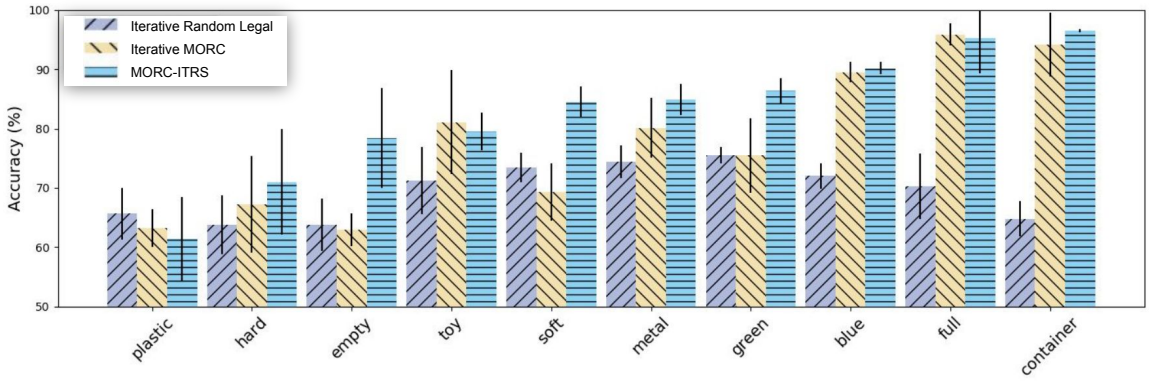
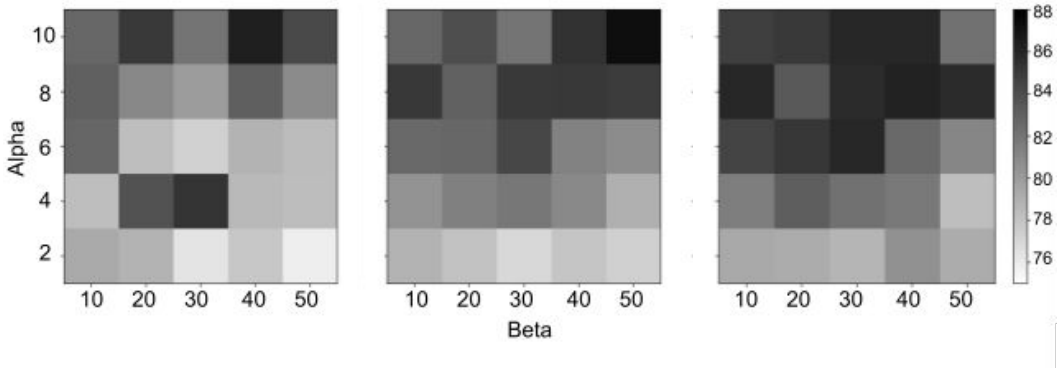


Figure 3.11: Accuracy of attribute identification tasks. The attributes (x-axis) are ranked based on MORC-ITRS' performance. MORC-ITRS performed the best on seven out of the ten attributes.



(a) Early (b) Middle (c) Late

Figure 3.12: We empirically evaluated the identification accuracies of MORC-ITRS in early (a), middle (b), and late (c) learning phase using different values of α and β, which are two parameters of our reward shaping approach (Eqn. 3.11).

in the beginning, but the accuracy does not improve much when it reaches the middle or late learning phase. A lower β value encourages the robot to explore no matter whether it is experienced or not, while a higher β value affects the robot to compute the optimal policy. Thus, when β is within the middle range, the robot has the best identification performance. We leave the auto-learning of the parameters to future work. Another observation is that the overall accuracy becomes higher in the middle (Middle) and late (Right) learning phases than early (Left) learning phase, which is expected and verifies the robustness of MORC-

ITRS to α and β selections.

3.7.4 Real Robot Demonstration of MORC-ITRS

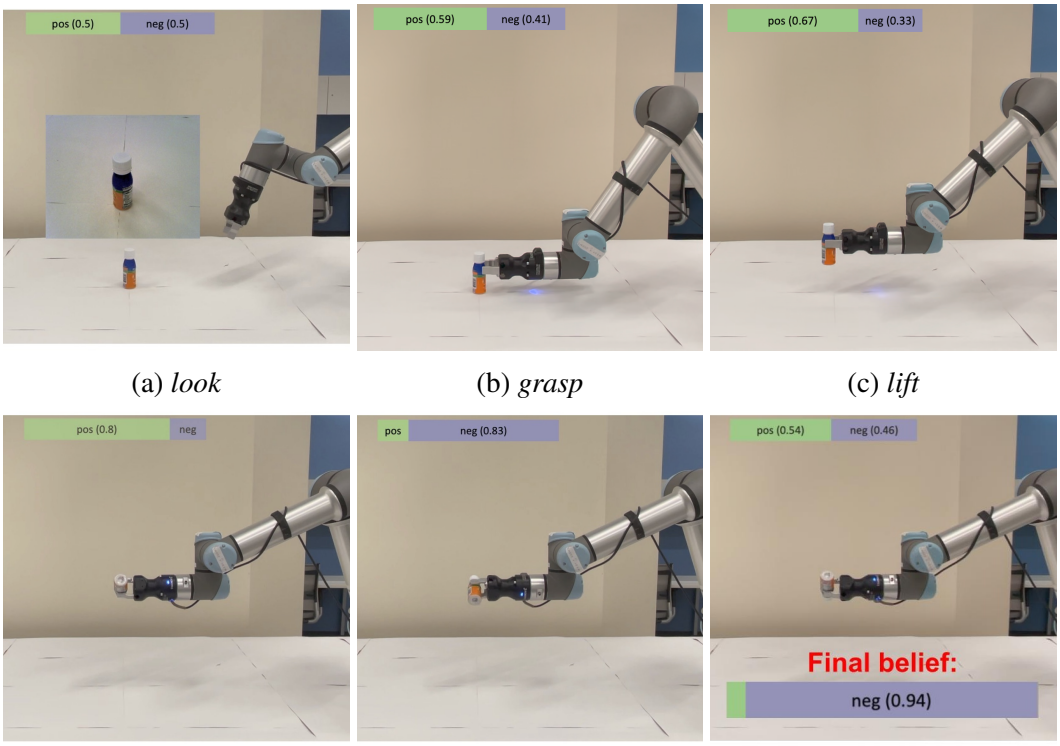
Table 3.5: Behaviors, observations, and belief updates in the demonstration trial of MORC-ITRS.

Step	Behavior	Observation	Belief (Initial belief: [0.5, 0.5])
1	*look*	*pos*	[0.41, 0.59]
2	*grasp*	*pos*	[0.33, 0.67]
3	*lift*	*pos*	[0.20, 0.80]
4	*shake*	*neg*	[0.83, 0.17]
5	*shake*	*pos*	[0.46, 0.54]
6	*shake*	*neg*	[0.94, 0.06]

We have demonstrated the learned action policy using a real robot (UR5e arm from Universal Robots). It should be noted that the three datasets we used in this research were collected on robots that are different from the robot in the demonstration. It is a major challenge in robotics of transferring skills learned from one robot to another. To alleviate the effect caused by the heterogeneity of robot platforms, after performing each action, we sampled a data instance from CY101 according to $x \in \mathscr{X}$, the fully observable component of the current state.

In the demonstration trial, our robot was given an object – a pill bottle half-full of beans. The one-attribute query was "*Is this object* EMPTY?" The robot performed a sequence of exploratory behaviors, as shown in Table 3.5, where we also listed the observation and the belief after each behavior. For instance, a *pos* observation means that the robot perceives that the object is EMPTY. Figure 5.6 shows a sequence of screenshots of the UR5e robot completing the task using a learned action policy.

Note that at step 3 in the demonstrated trial, lift was not very useful for identifying whether the object was empty or not. That was because the training set contained objects



(a) *look* (b) *grasp* (c) *lift*

(d) *shake* (e) *shake* (f) *shake*

Figure 3.13: A demonstration of the learned action policy. The robot performed six actions in a row. In the beginning, the robot started with a uniform distribution (it evenly believed the object can be EMPTY or not). After completing the six actions, the belief converged to "negative" (0.94 probability). Finally, the robot selected a reporting action to report that the object is not EMPTY."

that were of various weights causing a single *lift* action that could not distinguish between attributes such as EMPTY and LIGHT-WEIGHTED. In contrast, the *shake* actions produced a distinctive sound that seemed to be beneficial for leading the robot to a correct identification.

There were drastic changes in the belief after the two shake actions in Steps 5 and 6 in Table 3.5. At step 5, shake was executed successfully and led the system to the next state. However, there was uncertainty in the perceived sensory data which caused inaccurate outputs from the classifier and undesired belief changes from the robot. Intuitively, how the

belief evolves over time depends on how the robot trusts its actions on different attributes. When the robot believes an action is useful for detecting an attribute, this action might cause a drastic change in its belief; otherwise, the beliefs before and after the action might look similar. In this example, the robot believed *shake* is useful for detecting EMPTY, so the belief updates were significant after each of the three consecutive *shake* behaviors.

3.8 Conclusion

In this chapter, we introduce two Multimodal Embodied Attribute Learning (MEAL) problems that both require a robot to compute a policy of leveraging multimodal exploratory behaviors to identify object attributes. In OFFLINE-MEAL problems, the robot is provided data for learning action-conditioned attribute classifiers, whereas the robot does not have such data in ONLINE-MEAL domains. Accordingly, we have developed two algorithms called mixed observability robot control (MORC) and MORC with information-theoretic reward shaping (MORC-ITRS) for addressing OFFLINE- and ONLINE-MEAL problems respectively.

MORC uses mixed observability Markov decision processes (MOMDPs) to solve OFFLINE-MEAL problems, where a robot selects actions for multimodal perception in object exploration tasks. Our approach can dynamically construct a MOMDP model given an object description from a human user, compute a high-quality policy for this model, and use the policy to guide robot behaviors (such as *look* and *shake*) toward maximizing information gain. The dynamically built models in MORC enable the robot to focus on a minimum set of domain variables that are relevant to the current object and query. Attribute classifiers in MORC are learned using existing datasets collected with robots interacting with objects

in the real world. Experimental results show that MORC enables the robot to identify object attributes more accurately without introducing extra cost from exploratory behaviors compared to a baseline that suggests actions following a predefined action sequence.

MORC-ITRS selects exploratory behaviors toward simultaneous attribute classification and attribute identification. This algorithm is built on MORC, and provoides an information-theoretic reward function for the exploration-exploitation trade off in ONLINE-MEAL problems. The proposed method and baseline methods are evaluated using three real-world datasets. Experimental results show that MORC-ITRS enables the robot to complete attribute identification tasks at a higher accuracy using the same amount of training time compared to baselines.

4 Grounded Robot Task and Motion Planning

4.1 Introduction

Task and motion planning (TAMP) algorithms and systems have been used for robot planning at both discrete and continuous levels [91, 1]. Task planners sequence symbolic actions for guiding the robot's high-level behaviors [92], and motion planners calculate low-level motion trajectories in continuous spaces [93]. TAMP algorithms aim to bridge the gap between task planning and motion planning towards enabling robots to fulfill task-level goals and maintain motion-level feasibility at the same time [94, 95, 96, 97, 98, 99, 100, 101, 102].

One way to categorize TAMP domains is based on if a problem domain requires robot actions that take relatively short time (e.g., seconds, such as picking up and putting down objects) or relatively long time (e.g., minutes or even hours as navigating from one location to another) [103]. In the former type of domains, action feasibility is much more important to consider than plan efficiency, since extra plan steps do not add much time to the expected execution time. On the other hand, this paper is motivated by the latter type of TAMP domains, wherein it is advantageous to incorporate both *efficiency* and *feasibility* into the evaluation of plan qualities. Some existing TAMP research incorporates both efficiency and feasibility into task-motion planning [103, 98]. However, those methods evaluate feasibility

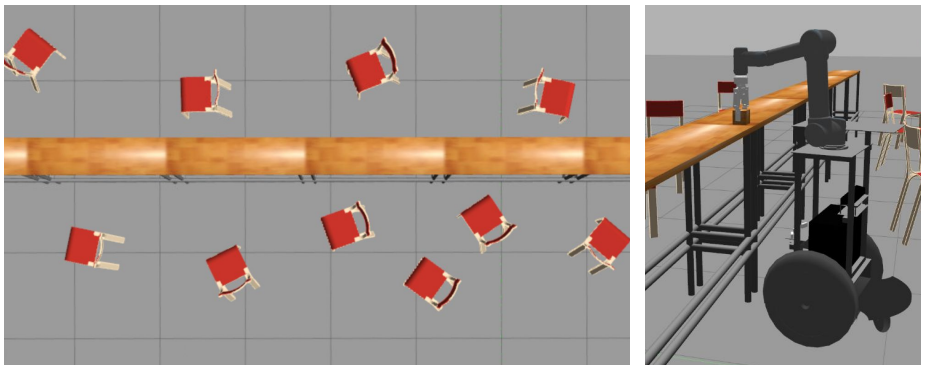


Figure 4.1: Our mobile manipulation domain that includes a long banquet table surrounded by chairs. Given a target location (on the table) to place an object, the robot needs to navigate to a location from which it can successfully perform the manipulation action, ideally as quickly as possible (thus preferring the near side of the table when feasible).

in a deterministic way, and rely on predefined "state mapping functions" for mapping each symbolic state into feasible poses in continuous space. For instance, to unload an object to a table, a robot needs to move to the table first, i.e., `beside(table)=true`, where previous TAMP research that we are aware of relies on predefined feasible poses that are spatially close to the table to evaluate the truthfulness of the "beside table" statement.

Such predefined state mapping functions that assume deterministic action feasibility have at least two deficiencies. First, a predefined state mapping function is not robust to dynamic obstacles (e.g., people seated around the table). Second, not all "feasible" behaviors are equally preferred, e.g., standing far and stretching out to place an object may be less preferred than standing close to do so. Those observations motivate this work that learns to evaluate action feasibility for robot task-motion planning.

The main contribution of this work is a visually grounded TAMP algorithm, called **G**rounded **RO**bot Task and Motion **P**lanning (GROP), that probabilistically evaluates action feasibility, and incorporates both feasibility and efficiency towards maximizing long-

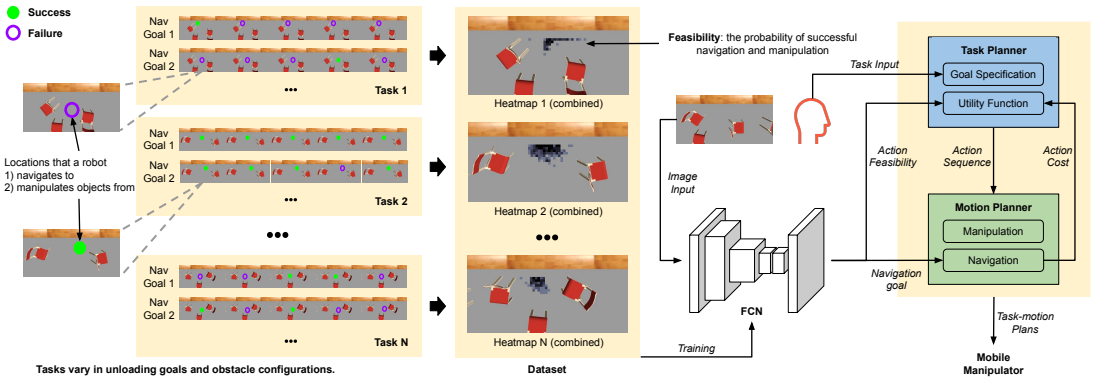


Figure 4.2: An overview of this work, including an FCN-based feasibility evaluation approach, and GROP, our grounded TAMP algorithm. A *task* corresponds to one "unloading goal" on the table, as well as a configuration of obstacles (chairs in our case). Given a task, every pixel is considered a navigation goal – the robot attempts to navigate there, and unload an object from there. This navigation-manipulation process is referred to as a *trial*. The robot performs multiple trials for each navigation goal, which yields a *feasibility* value for that particular location. The feasibility values together form one *heatmap* for each task. In our *dataset*, each instance is a top-down view image, whose label is the corresponding heatmap. The "Dataset" box shows a few "combined heatmaps" where heatmaps are overlaid onto the corresponding images. Training with the dataset generates an FCN that is used for two purposes: 1) evaluating the feasibility of task-level actions, and 2) selecting motion-level navigation goals. Finally, GROP incorporates both efficiency (measured by action costs) and feasibility to compute task-motion plans for a mobile manipulator.

term utility. Inspired by the concept of "symbol grounding" [6], we use "visual grounding" to refer to methods that use computer vision techniques to help an agent interpret abstract symbol tokens and connect them to the real world.

We have applied GROP to a domain of a mobile manipulator setting "dinner tables," as illustrated in Fig. 4.1. The robot needs to decide how to approach a table at the task level (e.g., from which side of the table), compute 2D navigation goals (connecting task and motion levels), and plan motion trajectories for navigation and manipulation behaviors. We have collected a dataset that includes $96,000$ instances of a robot conducting mobile manipulation tasks where in each instance, a robot unloads an object with dynamic obstacles surrounding a table. An instance is labeled "successful" if the robot is able to compute and

execute a task-motion plan that includes both navigation and manipulation actions. We use fully convolutional networks (FCNs) [12] to learn to visually ground spatial relationships and evaluate action feasibility. GROP is summarized in Fig. 4.2.

Compared with baselines from the literature [103, 104], GROP performed better in success rate while maintaining lower (or comparable) cumulative action costs. Finally, we demonstrate GROP with real-world robot hardware.

4.2 Related Work

TAMP methods aim to compute plans that fulfill task-level goals while maintaining motion-level feasibility, as reviewed in recent articles [91, 1]. Several TAMP algorithms have been introduced in recent years (e.g., [105, 106, 107, 108, 109, 110, 96, 111, 112, 113, 17, 95, 114, 115]. Within the TAMP context, we distinguish a few subareas of TAMP that are closest to this research on learning to visually ground symbolic spatial relationships towards planning efficient and feasible task-motion behaviors under uncertainty.

4.2.1 TAMP for Efficient and Feasible Behaviors

When high-level actions only take a few seconds, TAMP algorithms can focus mostly on action feasibility constraints without fully optimizing high-level plan efficiency. However, when there are actions that take significant time to execute (e.g., long-distance navigation), task-completion efficiency cannot be overlooked. Some recent methods have considered efficiency in different aspects of TAMP, such as planning task-level optimal behaviors in navigation domains [103], integrating reinforcement learning with symbolic planning in dynamic environments [97], computing safe and efficient plans for urban driving [16], and

optimizing robot navigation actions under the uncertainty from motion and sensing [98]. In contrast to those methods that do not have a perception component, GROP visually grounds symbols (about spatial relationships) to probabilistically evaluate action feasibility for task-motion planning.

4.2.2 TAMP under Uncertainty

While most TAMP methods assume a fully observable and deterministic world [1], some have been developed to account for the uncertainty from perception and action outcomes [11, 116, 117, 118, 119, 120]. For instance, the work of Kaelbling and Lozano-Pérez extended the "hierarchical planning in the now" approach to address both current-state uncertainty and future-state uncertainty [11]. Going beyond those methods that aim to maintain plan feasibility to complete tasks under high-level uncertainty, we consider uncertainty in the robot motion and also incorporate task-completion efficiency into the optimization of robot behaviors. As a result, our GROP algorithm is particularly suitable for TAMP domains that require robot operations over extended periods of time, such as long-distance navigation.

4.2.3 TAMP with Visual Perception

Recently developed methods have shown that visual information can be used to help robots predict plan feasibility, including task-level feasibility [121, 95], and motion-level feasibility [104, 99]. Those methods were developed to maximize task completion rate in manipulation domains, and actions that take relatively long time (such as long-distance navigation) were not included in their evaluations. Focusing on robots that operate over

extended periods of time, GROP (ours) incorporates efficiency into plan optimization. For instance, when highly feasible plans have very high costs, GROP supports the flexibility of executing slightly less feasible plans with much lower costs. GROP achieves this desirable trade-off between feasibility and efficiency by probabilistically evaluating plan feasibility, which is not supported by the above-mentioned methods.

4.3 Problem Statement

We consider a mobile manipulation domain that includes N objects Obj. There are obstacles (tables and chairs in our case) that prevent the robot from navigating to some positions in the domain. Location l is a symbolic concept that corresponds to a set of obstacle-free 2D poses (X), where each pose ($x \in X$) specifies a 2D position and an orientation. The robot needs to move each object $o \in Obj$ from its initial location to a goal position.

Actions: The robot is equipped with skills of performing a set of symbolic (task-level) actions denoted as $A : A^n \cup A^m$, where A^n and A^m are *navigation* actions and *manipulation* actions respectively. A navigation action $a^n_{l,l'} \in A^n$ is specified by its initial and goal locations, $l, l' \in L$, where L includes a set of symbolic locations. A manipulation action, $a^m_{o,l} \in A^m$, is specified by an object to be manipulated, $o \in Obj$, and a symbolic location, $l \in L$, to which the robot navigates and performs the manipulation action. We consider two types of manipulation actions of loading and unloading, represented by a^{m+} and a^{m-} respectively. Actions are defined via preconditions and effects. For instance, the action `load(`o_1`)` has preconditions of `at(robot,`l_1`)` and `at(`o_1`,`l_1`)`, meaning that to load the object o_1, the object must be co-located with the robot at the location l_1. The effects of

`load(`o_1`)` include o_1 being moved into the robot's hand, i.e., `inhand(`o_1`)`.

Perception: The robot visually perceives the environment through top-down views over the areas where manipulation and navigation actions are performed. We use *IM* to represent a 2D image that captures the current obstacle configuration, as shown in the "Image Input" of Fig. 4.2 (bottom right). To facilitate robot learning, we provide a dataset (as illustrated in the "Dataset" box of Fig. 4.2). Each instance includes a top-down view image, and a target object with a predefined position, while each label is in the form of a heatmap. Each pixel of a heatmap is associated with a 2D position, and has a "feasibility" value that represents the success rate of the robot navigating to the 2D position, and manipulating the target object from there.

A map is generated in a pre-processing step, and provided to the robot as prior information for navigation purposes using rangefinder sensors.

Uncertainty: The outcome of performing navigation action $a^n_{l,l'}$ to goal pose x is deterministic at the task level, but is non-deterministic at the motion level. In other words, the robot will end up in position x', which is not necessarily the same as x. This setting captures the fact that a mobile robot never achieves its exact 2D navigation goal (due to its imperfect localization and actuation capabilities), though successfully navigating to an area (l) is generally possible.

We focus on the interdependency between navigation and manipulation actions. For instance, the execution-time uncertainty from navigation actions results in different standing positions of the robot, which makes the outcomes of manipulation actions non-deterministic. This challenge generally exists in mobile manipulators. We assume no noise in the execution of manipulation actions (loading and unloading) to objects within a reachable area.

Format of Solution: A solution is in the form of a task-motion plan $p = \langle p^t, p^m \rangle$, where task plan p^t is of the form $\langle a_0^n, a_0^m, a_1^n, a_1^m, ... \rangle$, indicating that navigation and manipulation actions are interleaved. Motion plan p^m is of the form $\langle \xi_0^n, \xi_0^m, \xi_1^n, \xi_1^m, ... \rangle$, and ξ_i^n (or ξ_i^m) is a trajectory in continuous space for implementing symbolic action a_i^n (or a_i^m). The quality of task-motion plan p is evaluated using a utility function $\mathscr{U}(p)$, which considers both feasibility and efficiency of plan p:

$$\mathscr{U}(p) = \mathscr{R} \cdot \mathscr{F}(p) - \mathscr{C}(p), \tag{4.1}$$

where $\mathscr{F}(p) \in [0,1]$ is the plan feasibility (i.e., the probability that p can be successfully executed), $\mathscr{C}(p)$ is the overall plan cost of executing p, and $\mathscr{R} \rightarrow \mathbb{R}$ is a success bonus reflecting the reward from a successful execution. An optimal algorithm reports a task-motion plan of the highest utility:

$$p^* = \arg\max_p \mathscr{U}(p)$$

Remark: GROP agents are developed under the following assumptions that are aligned with the problem definitions presented in this section:

- the robot state space is predefined.
- actions and action realizations are unknown. Navigation behaviors are particular unreliable due to dynamic obstacles in cluttered environments.
- top-down views are available as part of the observation, so as lidar scan and a pre-built map (not including dynamic obstacles).

- the goal is to maximize success rate and minimize the total action cost of the current task.

Next, we present an algorithm that computes such task-motion plans through visually grounding spatial relationships while considering both efficiency and feasibility.

4.4 The GROP Algorithm

In this section, we introduce the paper's main contribution, an algorithm called **G**rounded **RO**bot Task and Motion **P**lanning, or GROP for short.

4.4.1 Algorithm Description

Algorithm 3 presents the GROP algorithm. Implementing GROP requires a task planner $Plnr^t$, a motion planner $Plnr^m$, a success bonus $\mathscr{R}\rightarrow\mathbb{R}$, and a cost function Cst that evaluates the cost of any motion trajectory generated by $Plnr^m$. Inputs of GROP include a rule-based task description T, a robot initial 2D position x^{init}, and a provided dataset D. GROP outputs a task-motion plan p in the form of $\langle p^t, p^m \rangle$.

GROP starts with training an FCN-based feasibility evaluator Ψ using provided dataset D in Line 2. Then it initializes an empty set of task-motion plans $\mathbf{P}$ in Line 3. $Plnr^t$ takes T as input and outputs a set of task-level satisficing plans, denoted as $\mathbf{P}^t$ in Line 4. The outer for-loop (Lines 5-22) iterates over each task-level satisficing plan. In each iteration, GROP evaluates the utility value of one task plan $\mathscr{U}(p)$, which incorporates both plan feasibility $\mathscr{F}(p)$ and plan efficiency $\mathscr{C}(p)$. Aiming to evaluate $\mathscr{F}(p)$ and $\mathscr{C}(p)$, each iteration in the first inner for-loop (Lines 8-14) considers a pair of navigation and manipulation actions in the task plan, and evaluates its feasibility and cost. In the second inner for-loop of Lines 15-

Algorithm 3 GROP

Require: Task planner $Plnr^t$, motion planner $Plnr^m$, success bonus $\mathscr{R}$, and cost function Cst
Input: Task description T, robot initial position x^{init}, dataset D
2: Train a motion-level feasibility evaluator Ψ using dataset D (detailed in Section 4.4.2)
3: Initialize a set of task-motion plans $\mathbf{P} \leftarrow \emptyset$
4: Compute a set of task-level satisficing plans: $\mathbf{P}^t \leftarrow Plnr^t(T)$
5: **for** each plan $p^t \in \mathbf{P}^t$ **do**
6: Initialize a motion-level position sequence: $X^{seq} \leftarrow [x^{init}]$
7: Initialize $tmp^f \leftarrow 0$ and $tmp^c \leftarrow 0$
8: **for** each action pair $\langle a^n_{l,l'}, a^m_{o,l'} \rangle$ in p^t **do**
9: Capture IM of location l'
10: Predict heatmap $h = \Psi(IM)$, using Eqn. 4.3
11: $tmp^f \leftarrow tmp^f + Fea^t(a^n_{l,l'}, a^m_{o,l'})$, using Eqn. 4.4
12: $x' \leftarrow Smp(l', h)$, and append x' to X^{seq}
13: $tmp^c \leftarrow tmp^c + Cst\big(Plnr^m(a^n_{l,l'})\big) + Cst\big(Plnr^m(a^m_{o,l'})\big)$
14: **end for**
15: **for** each $(x_i, x_{i+1}) \in X^{seq}$ **do**
16: Compute motion-level trajectory $\xi \leftarrow P^m(x_i, x_{i+1})$
17: Append ξ to motion plan p^m
18: **end for**
19: Generate task-motion plan $p \leftarrow \langle p^t, p^m \rangle$, and append p to the task-motion plan set $\mathbf{P}$
20: Update $\mathscr{F}(p) \leftarrow tmp^f$ and $\mathscr{C}(p) \leftarrow tmp^c$
21: $\mathscr{U}(p) \leftarrow \mathscr{R} \cdot \mathscr{F}(p) - \mathscr{C}(p)$ (Eqn. 4.1)
22: **end for**
23: Compute optimal task-motion plan: $p^* = \arg\max_{p \in \mathbf{P}} \mathscr{U}(p)$
return p^*

18, GROP calls $Plnr^m$ to compute one motion plan for each task-level action. Line 19 puts together task plan p^t and motion plan p^m to form a task-motion plan p. In the same line, p is added to task-motion plan set $\mathbf{P}$. Lines 23-23 are the final steps to select and return the optimal task-motion plan from $\mathbf{P}$ given utility function $\mathscr{U}(p)$.

4.4.2 Feasibility Evaluation

In this subsection, we discuss how to evaluate action feasibility at task and motion levels (Line 11 in Algorithm 3), where the feasibility evaluation at the task level relies on the feasibility evaluation at the motion level.

Motion-Level Feasibility: In our mobile manipulation domain, motion-level feasibility

$Fea^m(x,y)$ is a function of 2D positions x and y, and is the probability of a robot successfully navigating to x and manipulating an object that is in position y. $Fea^m(x,y)$ can be extracted from gray-scale heatmap image h^y that is centered around y:

$$Fea^m(x,y) = h^y[x] \tag{4.2}$$

We use a FCN-based feasibility evaluator Ψ to generate heatmap h^y, given a top-down view image IM^y captured right above unloading position y ("Image Input" in Fig. 4.2):

$$h^y = \Psi(IM^y) \tag{4.3}$$

Data Collection and Learning Ψ with FCN: Here we discuss how to learn Ψ in Equation 4.3. A *task* specifies an obstacle configuration and a position y that a robot wants to unload objects to. In each *trial* of our data collection process, a robot attempts to navigate to position x, and then unload an object to position y. Such a trial produces a data point in the following format:

$$(IM^y, x) : r$$

where IM^y is a top-down view image captured right above y, and r is either *true* or *false* depending on if the robot succeeds in both navigation and manipulation actions. The robot repeated the same process for N times ($N = 5$ in our case), and we used the results $(r_0, r_1, \cdots, r_{N-1})$ to compute a success rate for positions x and y, which determines a gray-scale color for one pixel of a heatmap: $h[x]$.

Iterating over all possible positions of x in an area of $Width \times Height$ (24 pixels by 8

pixels in our case) in image *IM*, we were able to generate one full heatmap h for the current task. Here we assume this area is large enough to cover all positions, from which the robot can unload objects to y. To diversify the instances, we randomly placed obstacles (chairs in our case) to generate ten different "environments," and then randomly sampled unloading positions to generate a total of 100 tasks. As a result, our dataset contains 100 instances, each in the form of a top-down view image (64×32). Each instance has a label that is in the form of a heatmap. The size of our dataset is $96,000$, i.e., $100 \times N \times Width \times Height$.

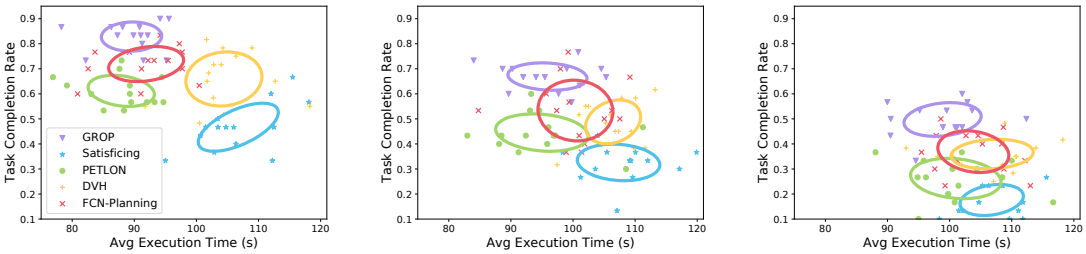
Task-Level Feasibility: $Fea^t(a^n_{l,l'}, a^m_{o,l'})$ evaluates the feasibility (in the form of a probability) of a robot successfully performing both task-level navigation action $a^n_{l,l'}$ and task-level manipulation action $a^m_{o,l'}$.

$$Fea^t(a^n_{l,l'}, a^m_{o,l'}) = \frac{\sum_{i=0\cdots N-1} Fea^m\big(Smp_i(l',h), y\big)}{N} \tag{4.4}$$

where function $Smp_i(l',h)$ samples the ith 2D position from location l'. The positions are weighted by heatmap h that is centered around object o. Intuitively, positions of higher motion-level feasibility are more likely to be sampled.

4.5 Experiments

We conducted extensive experiments in simulation, where a mobile manipulator performs navigation and manipulation actions to set "dinner table." We also demonstrate GROP using a real robot system that includes a mobile platform and a robot arm. Our main hypothesis is that GROP outperforms existing TAMP algorithms in task completion rate without introducing additional action costs.



(a) Easy (b) Normal (c) Hard

Figure 4.3: Overall performances of GROP and four baseline methods in efficiency (x-axis) and task completion rate (y-axis). Tasks are grouped based on their difficulties. The ellipses represent the means and 2D standard variances of each approach. GROP produced the highest task completion rate, while maintaining smaller or comparable execution time. This observation is consistent over tasks of different difficulties.

Baselines: GROP is evaluated through comparisons with the following baselines. All baselines are TAMP algorithms, and they vary in whether efficiency is considered in plan optimization, and whether feasibility is considered. All baselines select navigation goals by randomly sampling an obstacle-free position that is close to the unloading position.

- **Satisficing** (weakest baseline): Action costs are not considered, so it does not avoid long-distance navigation. All actions share the same feasibility (FCN not used).
- **PETLON** [103]: It considers plan efficiency, but does not quantitatively evaluate action feasibility.
- **DVH** [104]: It does not consider plan efficiency, but quantitatively evaluates action feasibility.
- **FCN-Planning** (most competitive): The same as GROP except that the heatmap (Line 12 in Algorithm 3) is not used for selecting 2D navigation goals.

It should be noted that, we cannot authentically implement DVH [104] for evaluation, because they used convolutional neural networks (CNNs) for task-level action feasibility

evaluation, and we do not have a dataset from our domain for training the CNNs. We did the best we could by replacing their CNN-based visual component with our FCN-based feasibility evaluator.

Experiment Setup: The mobile manipulator includes a UR5e robot arm, a Robotiq 2F-140 gripper, an RMP 110 mobile base, and a Velodyne VLP-16 lidar sensor. We used the Building-Wide Intelligence (BWI) codebase [27] to construct our simulation platform, which relies on the Gazebo physics engine [29]. We use a Rapidly exploring Random Tree (RRT) approach [24] to compute motion-level manipulation plans. The navigation stack was built using the `move_base` package of Robot Operating System (ROS) [26]. The robot's task planner is ASP-based [22, 23] and we used the Clingo solver for computing task plans [122].

The dataset described in Section 4.4.2 was fed into an FCN for training Ψ. We adapted the FCN-VGG16 model [12] and trained it with batch size 4 and learning rate e^{-3}. We used a machine equipped with an Intel 3.80GHz i7-10700k CPU and a GeForce RTX 3070 GPU on a Ubuntu system.

The test environment contains two tables, one for loading and the other (a long banquet table) for unloading. Obstacles (chairs) are randomly placed near the unloading table. Positions and the number of chairs are dynamically changed for different environments. An RGB camera is attached to the ceiling to capture overhead images of environments. A mobile manipulator is tasked with moving three objects from the loading table to three different positions on the unloading table, where the robot can hold multiple objects at the same time. There is a tolerance of $0.1m$ for unloading actions, and an unloading action is considered unsuccessful if the object is more than this distance away from the specified

Table 4.1: Task completion rate / average execution time in one of the environments with different robot's navigation velocities.

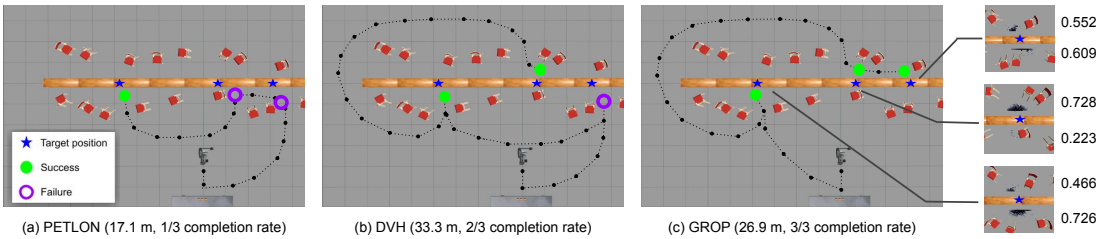
	GROP	*PETLON*	*DVH*
Slow	0.80 / 166.16	0.63 / 166.46	0.73 / 204.04
Medium	0.82 / 95.42	0.63 / 93.23	0.73 / 112.02
Fast	0.88 / 59.56	0.63 / 56.62	0.73 / 66.01

unloading position. Task completion is evaluated based on whether each "seat" of the table is set up. Reward $\mathscr{R}$ has a value of 40 in our utility function defined in Equation 4.1.

GROP vs. Baselines: Fig. 4.3 shows the main results from experiments of comparing GROP to the four baselines. There were a total of 420 different tasks in 30 different environments. Each data point in the figure represents an average of 10 tasks. We grouped the tasks based on their difficulties: Easy, Normal, and Hard. A task's *difficulty* is measured by the total area that a robot can navigate to and unload an object from. For instance, a task with all unloading positions being surrounded by obstacles has a high difficulty. After sorting the tasks based on their difficulties, we evenly placed them into the three groups.

GROP consistently performed better in task completion rate (y-axis) in all three settings, while maintaining high plan efficiency (x-axis). We also see that GROP performed particularly well in hard tasks where it produced the highest completion rate and the lowest action costs. While PETLON generated efficient plans (comparable to GROP), it does not reason about feasibility, resulting in low completion rate. DVH generates feasible plans (like FCN-Planning), but it does not consider action costs, resulting in long execution time in task completions. Results support our hypothesis that GROP improves plan efficiency without introducing additional action costs.

Robot Velocities: In this experiment, we used three robots that move at different velocities: 0.2 m/s (*Slow*), 0.4 m/s (*Normal*), and 0.8 m/s (*Fast*). Results are shown in Table 4.1.



(a) PETLON (17.1 m, 1/3 completion rate) (b) DVH (33.3 m, 2/3 completion rate) (c) GROP (26.9 m, 3/3 completion rate)

Figure 4.4: Three illustrative trials using GROP and two baselines (PETLON and DVH). The robot needs to move three objects from the loading table (bottom) to three unloading target positions marked by blue stars, where the robot can hold multiple objects. Green dots (or purple circles) represent a robot successfully (or unsuccessfully) navigating to the position and unloading an object to the corresponding target position. Three heatmaps are overlaid onto overhead images, as shown on the right, indicating the feasibility values of navigating to and unloading from different positions. The numbers on the very right represent task-level action feasibility values of unloading from one side of the table. Under each subfigure, we present the total navigation distance and task completion rate, where we see GROP produced the highest completion rate, and performed better than DVH in efficiency.

Here we compare GROP to only the two baselines that are available from the literature (PETLON and DVH). We see that GROP outperforms the two baselines in task completion rate. What is interesting is that when the robot moves fast, GROP automatically weighted feasibility more, because the robot will not take too long to complete a navigation task anyway. As a result, GROP produced the highest task completion rate of 0.88 on a fast robot, while the baselines are not adaptive to the robot's velocity.

Illustrative Trials in Simulation: Fig. 4.4 shows three illustrative trials using GROP (ours) and two baselines (PETLON and DVH), where GROP produced the highest completion rate (3/3), while the baselines succeeded in at most two tasks. PETLON does not evaluate plan feasibility, and planned to unload objects to the middle and right positions from the south. In particular, unloading to the middle unloading position from the south is very difficult (with a feasibility value of 0.223). PETLON does not take such factors

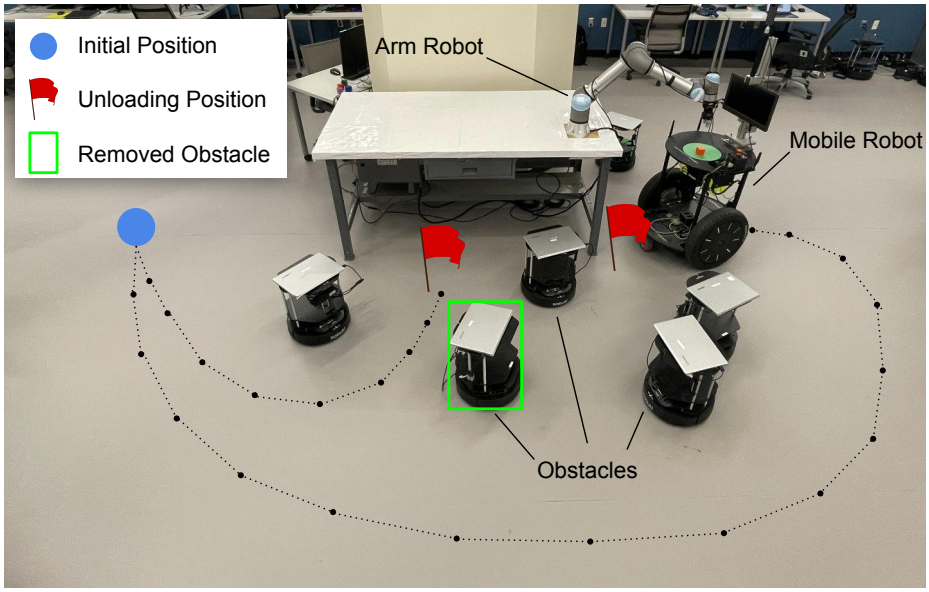


Figure 4.5: The arm robot is placing an object onto the mobile robot in trial T_1. There are two loading positions on the south and east sides of the table, marked by red flags. The mobile robot's initial position is shown as the blue dot. The green box highlights the obstacle that was removed in T_2.

into consideration, which produced failures in unloading to the middle position. DVH does not consider efficiency in plan optimization, and generated a plan with long-distance navigation actions. GROP incorporates both efficiency and feasibility, and produced the best overall performance.

Real Robot Demonstration: We demonstrate two trials of T_1 and T_2 using GROP on a real-robot platform. Instead of using a mobile manipulator, we used a robot system that includes two robots of a Segway-based mobile platform and a UR5e robot arm. The mobile robot started from an initial position, and was tasked with loading a distant object (an orange cube in our case) from the arm robot. The object was on the same table as the arm robot is, where the arm robot could pick the object, and place it onto the mobile robot to complete a loading behavior.

In trial T_1, the system computed the task-level feasibility values of loading from the south and east sides: 0.377 and 0.721. The corresponding costs were 7.5 and 19.3 respectively (distances of $4.5m$ and $11.6m$), where the robot moved at speed $0.6m/s$. GROP evaluated the utility values (7.6 and 9.5 in this case), and decided to load from the east side (less efficient but more feasible), as shown in Fig. 5.6.

In trial T_2, the robot system worked on the same task, while one obstacle (green box in Fig. 5.6) was removed from the environment. The obstacle removal changed the feasibility: Loading from the south has higher feasibility of 0.520, and a higher utility of 13.3. Accordingly, the mobile robot decided to load the object from the south, where there existed little chance of failing in the loading behavior but the overall efficiency was significantly improved. In both demonstration trials, the robot system succeeded in loading the object to the mobile platform.

4.6 Conclusion

This chapter introduces an algorithm, called **G**rounded **RO**bot Task and Motion **P**lanning (GROP), that considers both efficiency and feasibility for robot task-motion planning. GROP visually grounds spatial relationships to probabilistically evaluate action feasibility, and is particularly suitable for TAMP domains with long-term robot operations (e.g., long-distance navigation). We have extensively evaluated GROP in simulation using a mobile manipulator, and demonstrated it using a real robot system that includes a mobile robot and an arm robot. Results showed that GROP outperformed competitive baselines from the literature in plan efficiency without introducing additional action costs.

5 Symbolic State Space Optimization

5.1 Introduction

At the task level, robots frequently use symbolic planners to sequence high-level actions [92]. At the motion level, each high-level action is grounded to low-level trajectories in continuous spaces using motion planners [93]. TAMP algorithms aim to bridge the gap between task planning and motion planning towards enabling robots to fulfill task-level goals and maintain motion-level feasibility at the same time [91, 1]. A common and widely accepted assumption for most TAMP research is that the task planner is predefined by a domain expert who manually specifies a symbolic state space. In this paper, we discuss TAMP in a long horizon mobile manipulation domain where the robot is given a task of repeated navigation to perform manipulation behaviors (e.g., pick and place) in different places.

Nevertheless, manually constructing state spaces might not be desirable in some scenarios. Fig. 5.1 shows a situation where in long horizon mobile manipulation domains, if each object is placed at a separate symbolic location as defined in the task-level state space, the robot will always need to navigate before picking up the next object. This is because the task planner believes only a navigation action can bring the robot to the location required by the next manipulation action. In practice, however, the robot often picks

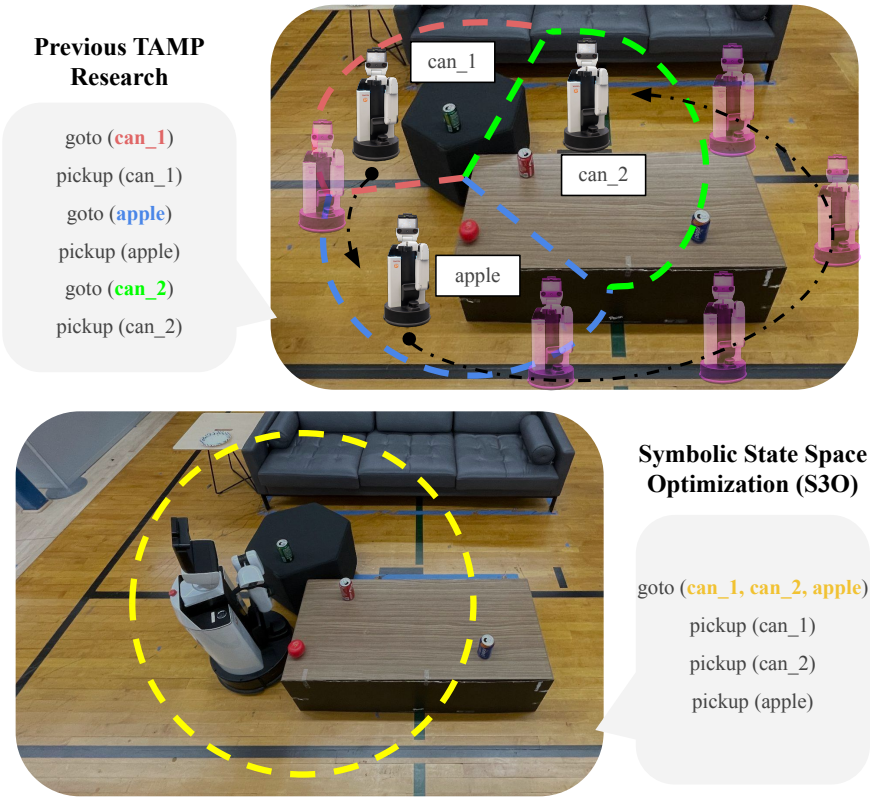


Figure 5.1: Objects are frequently in separate symbolic locations in a predefined task planner. A TAMP system with such a fine-grained state space would always generate plans that suggest the robot navigate before every manipulation. However, if an optimized state space can include multiple objects (that are close to each other) in a single location, the robot will be able to navigate once and perform a sequence of manipulation actions. We aim to answer how to compute such symbolic locations and their geometric groundings.

up multiple objects from a single position, for example, as restaurant waiters can easily identify a standing location that allows them to pick up multiple dishes at once. Especially when objects are located close to each other, it is unnecessary for the robot to navigate before every manipulation. This observation motivated the development of this research on optimizing symbolic state spaces for task planners to best facilitate TAMP for long horizon mobile manipulation.

One of the challenges in optimizing symbolic state spaces for TAMP, which is the focus

of this work, comes from the uncertainties in action and perception. We consider failures in navigation and manipulation behaviors, e.g., due to the robot being too close to obstacles or too far from the target objects. To this end, we propose **S**ymbolic **S**tate **S**pace **O**ptimization (S3O) based on probabilistically evaluated action feasibility under uncertainty. S3O partitions the continuous configuration space into a set of abstracted locations with their 2D geometric groundings to compute efficient and feasible task-motion plans in long horizon mobile manipulation domains.

Fig. 5.2 shows an overview of S3O which first constructs a candidate set of object-centric symbolic state spaces using Voronoi Partitioning [123]. Then the algorithm ranks each state space by a scoring function developed using feasibility evaluation from robot perception. The ranking mechanism effectively reduces the search complexity of state spaces by controlling the size of the candidate set. Top-ranked state spaces are used for constructing the task planner in the TAMP system where we further apply an Evolution Strategy (ES) algorithm [124] for efficient motion-level search. The proposed framework has been quantitatively evaluated in simulation and qualitatively demonstrated on real hardware, where the robot works on the task of "clearing up dining tables." Compared to existing TAMP baselines, experimental results show that our approach consistently leads to high-quality task-motion plans in terms of task completion rate and plan execution time.

5.2 Related Work

In this section, we describe the three most related research areas, including those task and motion planning methods that optimize plan efficiency and feasibility, research that learns symbolic representations for robot planning, and the application domain of long

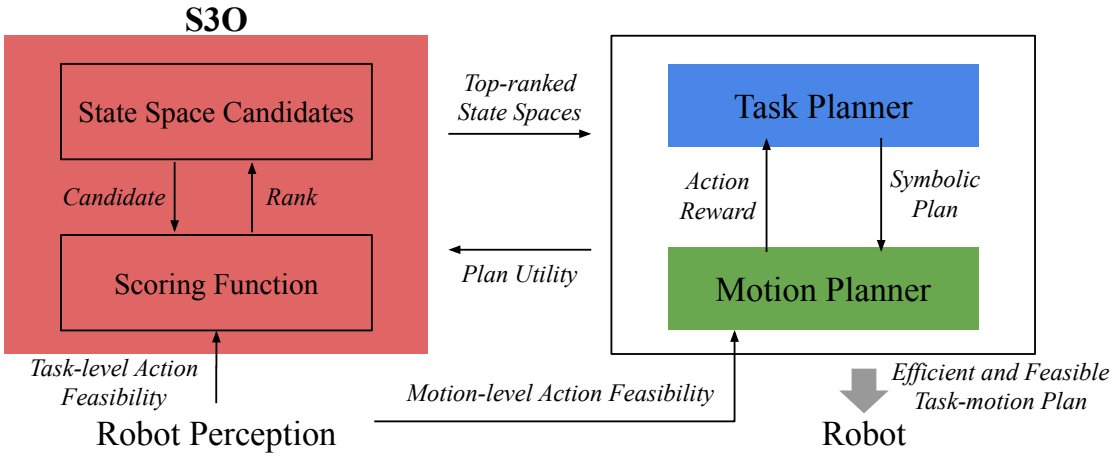


Figure 5.2: An overview of Symbolic State Space Optimization (S3O) for Task and Motion Planning systems.

horizon mobile manipulation.

5.2.1 TAMP for Efficient and Feasible Behaviors

Task and motion planning research can be categorized into two groups: one includes high-level actions that take no more than a few seconds (e.g., picking up, putting down and pushing objects), the other requires robot actions taking relatively long time (e.g., long-term navigation) [103]. The former type of TAMP has a long history in the literature and focuses mostly on action feasibility [94, 95, 96, 99, 100, 101, 102], while some recent methods have considered behavioral efficiency in the latter type of TAMP, usually in robot navigation, autonomous driving, or mobile manipulation domains [103, 97, 16, 98, 4, 125, 20]. One common assumption for these TAMP methods is the predefined task planner. Unlike those, we probabilistically compute action feasibility via visual perception to optimize the task-level state space.

5.2.2 Symbol Learning for Robot Planning

Learning-based methods have shown effectiveness in model acquisition and symbol generation for robot planning. Researchers have learned action preconditions and effects models for enabling purely symbolic planning [84] and integrated task-motion planning [126]. Some other work focuses on symbol learning and mapping, such as connecting natural language to learned symbolic abstractions [127], learning state abstractions for bootstrapping motion planning [128], and learning to ground the physical meanings of object attribute symbols in the real world [17]. In our work, we also learn to generate and map each symbol from the continuous space, but going beyond that, we further optimize the efficiency and feasibility of task-motion plans using the learned symbols.

5.2.3 Long Horizon Mobile Manipulation

There is rich literature on learning and planning coordinated actions for mobile manipulation [9, 10]. Most existing methods focused on positioning the base of a mobile manipulator in such a way that manipulability is maximized [129, 130, 131, 132]. A convincing technique is using robot reachability maps [133, 134]. Recent research applies reinforcement learning in a hierarchical style to tackle this problem [135, 136, 137, 138, 139]. In this paper, we not only consider coordinated navigation and manipulation, but also optimize a sequence of mobile manipulation actions over a long horizon. Sequential mobile manipulation tasks [140] have been studied, including works that aimed to minimize platform movements to reach a set of poses in the workspace [141] or to minimize the overall cost of completing the task [142]. As compared to our approach, we also consider

perception and assume execution-time uncertainty from both perception and actuation.

5.3 Problem Statement

We present the terminologies, assumptions, and objectives of the TAMP problem we focus on in this research: a long horizon mobile manipulation task where the robot repeatedly navigates and picks up multiple objects in different locations.

Symbols and Symbol Mapping: $\mathscr{O} = \{o_1, o_2...\}$ is a set of target objects that can be moved by a robot. $\mathscr{L} = \{l_1, l_2...\}$ is a set of symbolic locations. Let $y \in \mathscr{Y}$ be a set of xy poses in continuous space. $Sym : \mathscr{Y} \rightarrow \mathscr{L}$ is a function that maps any 2D geometric position $y \in \mathscr{Y}$ to a symbolic location $l \in \mathscr{L}$. The symbolic state space of our problem is defined in the form of $\langle \mathscr{L}, Sym, \mathscr{Y} \rangle$.

Actions: The robot is equipped with skills of performing a set of actions denoted as $\mathscr{A} : \mathscr{A}^n \cup \mathscr{A}^m$, where $\mathscr{A}^n$ and $\mathscr{A}^m$ are *navigation* actions and *manipulation* actions respectively. A navigation action $a^n : \langle l_r, l'_r, y_r, y'_r \rangle \in \mathscr{A}^n$ is specified at both low and high levels: 1) the robot's current and next symbolic locations that are denoted as $l_r, l'_r \in \mathscr{L}$; 2) the corresponding 2D coordinates y_r, y'_r mapped by *Sym*. *r* is a symbol to denote "robot" as being distinguished from symbol *o* for "object". A manipulation action $a^m : \langle o, l, y_r, y_o \rangle \in \mathscr{A}^m$ is specified by an object (i.e., o) to be manipulated, the object's 2D location, y_o, the object's and the robot's symbolic location, $l \in \mathscr{L}$. The robot and the object to be manipulated should be in the same symbolic location. In this work, we consider `pickup` as a manipulation action and `goto` as a navigation action. Actions are defined via preconditions and effects. For instance, the action `pickup(`o_1`)` has preconditions of `at(robot,` l_1`)` and

`at(`o_1`, `l_1`)`, meaning that to pick up the object o_1, the object must be co-located with the robot base in the same symbolic location l_1. The effects of `pickup(`o_1`)` include o_1 being moved into the robot's hand, i.e., `inhand(`o_1`)`.

Action Uncertainty: Let $\mathscr{T} : \mathscr{T}^n \cup \mathscr{T}^m$ be a set of probability distributions for modeling action uncertainties. For a navigation action, $\mathscr{T}^n(\hat{y}'_r|y_r, y'_r)$ represents the probability of a mobile robot aiming to navigate to goal y'_r, while landing in $\hat{y}'_r$, given the current robot position y_r. For a manipulation action, $\mathscr{T}^m(\hat{y}'_o|y_o, y_r)$ represents the probability of the robot given an end effector goal position y_o of "reaching" the object, while ending up at a position $\hat{y}'_o$, given the robot's standing position y_r. In practice, $\mathscr{T}^n$ and $\mathscr{T}^m$ are determined by the robot's navigation and manipulation systems. In this paper, both $\mathscr{T}^n$and $\mathscr{T}^m$ are treated as black box.

Perception: The robot visually perceives the environment. While we provide the robot with top-down view images in this work, our approach can be combined with perception methods that rely on first-person view for object pose estimation [143]. A map is generated in a pre-processing step, and provided to the robot as prior information for navigation purposes using rangefinder sensors. Please note that dynamic obstacles such as randomly-placed chairs are not in the map.

Problem Formulation: The input of the problem is a tuple $\langle \mathscr{Y}_o^{init}, y_r^{init}, \mathscr{A} \rangle$. $\mathscr{Y}_o^{init}$ is a set of objects' initial positions and y_r^{init} is the robot's initial base pose. The problem outputs a task-motion plan p which is in the form of a sequence of navigation actions $a^n \in A^n$ and manipulation actions $a^m \in A^m$. The problem finds a task planner $Pln^{\mathscr{L},Sym,\mathscr{Y}}$ that is parameterized by the symbolic state space $\langle L, Sym, \mathscr{Y} \rangle$, in order to compute a task-motion

plan p, where the objective is to maximize the plan utility for improving task completion rate and reducing robot execution time.

Remark: S3O agents are developed under the following assumptions that are aligned with the problem definitions presented in this section:

- the robot state space is not predefined, including unknown symbolic locations.
- actions and action realizations are unknown. Navigation behaviors are particular unreliable due to dynamic obstacles in cluttered environments.
- top-down views are available as part of the observation, so as lidar scan and a pre-built map (not including dynamic obstacles).
- the goal is to maximize success rate and minimize the total action cost of the current task.

5.4 Symbolic State Space Optimization (S3O)

In this section, we present the paper's main contribution called Symbolic State Space Optimization (S3O) which optimizes the state space for the task planner based on probabilistically evaluated action feasibility. S3O first constructs symbolic state spaces using object-centric Voronoi Partitioning and robot reachability. And then it ranks a set of candidate state spaces based on evaluated action feasibility.

Constructing Symbolic State Spaces: We construct state spaces following two principles: 1) states (i.e., locations) should be determined by which object(s) they are the closest to; 2) the distance from the object to each pose in a state should be within the maximum

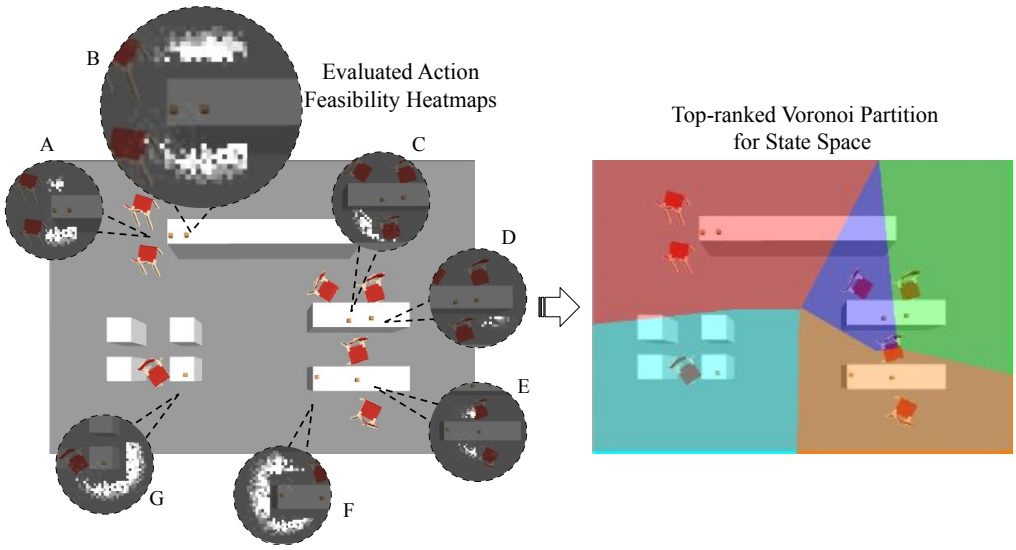


Figure 5.3: **Left:** Action feasibility values are computed using robot perception and represented as heatmaps. **Right:** A top-ranked Voronoi Partition for the state space generated using S3O, where objects A and B are in one symbolic location, and objects E and F are in another one.

reachability (1 meter in our case) of the robot. Thus, in our framework, we consider poses that are around the objects within 1 meter, and generate areas by object positions in the 2D configuration space using the Voronoi Partitioning algorithm. The distance from each 2D pose in an area to its corresponding object position is less than that from every other object position. Each area in the Voronoi diagram is considered as a symbolic location l, and the whole Voronoi partition corresponds to a set of locations $\mathscr{L}$ as well as a symbol mapping function Sym to map each 2D pose to a location $l \in \mathscr{L}$. Further, possible adjacency area merging operations are conducted in the Voronoi diagram. Each area merging that results in a new symbolic state space (i.e., $\langle \mathscr{L}, Sym, \mathscr{Y} \rangle$) is considered as a state space candidate.

Scoring Function for State Space Ranking: In order to deal with a large number of objects, we compute scores for each state space candidate, i.e., $\langle \mathscr{L}, Sym, \mathscr{Y} \rangle$. The score is

calculated using the following function that is based on action feasibility:

$$Score(\langle \mathscr{L}, Sym, \mathscr{Y} \rangle) = \sum_{o \in \mathscr{O}} Fea^t(l,o), \text{if } \texttt{at(}o\texttt{, } l\texttt{)} \tag{5.1}$$

where $Fea^t(l,o)$ is the task-level action feasibility function that computes the probability of the robot navigating to location l and picking up object o. Intuitively, if the symbolic state space has a high accumulative task-level feasibility value over all the objects, this state space will be evaluated with a high score. Fig. 5.3 shows the evaluated action feasibility (represented as heatmaps) and a top-ranked Voronoi Partition for the state space.

After ranking the state spaces by the scores computed using Eqn. 5.1, we select the top K state spaces to construct K task planners at robot planning time. In each TAMP search iteration, our system normalizes the scores to produce a probability distribution from which one of the task planners is chosen. The system plans in parallel, each with a sampled task planner, and uses arg max to find the state space (i.e., $\langle \mathscr{L}, Sym, \mathscr{Y} \rangle$) that generates a plan of the highest utility.

Action Feasibility Evaluation: Robot perception is used to probabilistically evaluate action feasibility, represented as function $Fea : Fea^t \cup Fea^m$. The task-level feasibility function $Fea^t(l,o)$ takes a symbolic location l and an object o as input, while the motion-level feasibility function $Fea^m(y_r, y_o)$ takes a robot 2D pose y^r and an object 2D pose y^o as input. Both task-level and motion-level feasibility functions output feasibility values ranging from 0.0 (infeasible) to 1.0 (feasible). In this work, Fea^t serves as a key component in the proposed scoring function (Eqn. 5.1). Fea^m is used to compute: 1) Fea^t, which is discussed in the next paragraph, and 2) the plan utility, which is formally defined in the

next section.

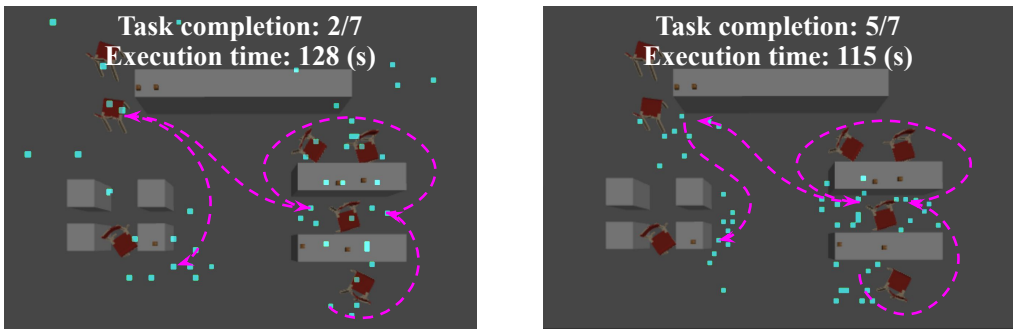
Our task-level feasibility function $Fea^t(l,o)$ shares the same definition as what was initially introduced in [4]. Briefly summarizing here, $Fea^t(l,o)$ relies on $Fea^m(y_r,y_o)$ and a sampling function Smp. $Fea^m(y_r,y_o)$ computes the motion-level feasibility of robot navigating to 2D position y_r and picking up the object that is at position y_o. Smp samples 2D positions y_r that satisfy $Sym(y_r) = l$, where the positions are weighted by $Fea^m(y_r,y_o)$. In other words, positions of higher motion-level feasibility are more likely to be sampled. Computing $Fea^t(l,o)$ is to calculate the average motion-level feasibility over N samples from Smp.

We extract $Fea^m(y_r,y_o)$ from a learned Fully Convolutional Network model [12], which is trained using robot data from past experience, represented as gray-scale heatmap images. We trained the model by collecting a dataset that diversifies the obstacle (i.e., chair) positions and is with randomly-placed objects on the table. One recent work uses the same architecture for motion-level feasibility evaluation [4], but their model can only deal with objects that are of a predefined distance from the table edge due to the limitation of its training dataset. In comparison, the motion-level feasibility function extracted from our model equips the robot with the capability of handling more generalized object pick and place tasks.

5.5 Computing Task-motion Plans

Long horizon mobile manipulation domains require robots to complete tasks as accurately and quickly as possible. This section details how S3O computes task-motion plans.

As described in Sec. 5.3, the objective of the problem is to maximize the overall task



(a) Early iteration.

(b) Late iteration.

Figure 5.4: Samples (cyan pixels) drawn from the CMA-ES sampler at early and late iterations. With action feasibility and efficiency being considered in the objective function, robot base positions gradually converge to a sequence of areas that are close to the objects for the robot to reach, and are of a low overall navigation cost.

completion rate and minimize the robot execution time. Robot execution time is largely affected by how much time each action takes (especially long-range navigation actions), and the task completion rate depends on manipulation action feasibility. To this end, at planning time, we design the cost function for an action a as:

$$Cst(a) = \begin{cases} len(y_r, y_r')/v + \gamma, \text{ if } a \in \mathscr{A}^n \\ \delta, \text{ if } a \in \mathscr{A}^m \end{cases} \tag{5.2}$$

where function len is able to measure the trajectory length of executing a navigation action and v is the robot speed. γ is a constant cost for navigation when the robot starts moving, which motivates the robot to select as few navigation actions as possible. δ is a constant cost for manipulation actions which is relatively small as compared to the cost for navigation actions.

We further use the action cost function to design the action reward function. Let λ be a

successful reward bonus of picking up an object. The reward function is as follows:

$$R(a) = \begin{cases} -Cst(a), \text{if } a \in \mathscr{A}^n \\ \\ -Cst(a) + Fea^m(y^r, y^o) \cdot \lambda, \text{if } a \in \mathscr{A}^m \end{cases} \tag{5.3}$$

We use the CMA-ES optimization technique [124] to serve as the sampling algorithm for motion-level 2D poses that the robot navigates to and performs the manipulation action(s) at. Fig. 5.4 shows an example of the samples drawn from early and late iterations of the CMA-ES sampler. Each sample we draw is in the form of $\langle y_r^1[x], y_r^1[y], y_r^2[x], y_r^2[y], ... \rangle$, where $y_r^i[x]$ ($y_r^i[y]$) denotes the x (y) coordinate of the robot pose for navigating to and picking up the ith object from. We maintain an independent CMA-ES sampler for each fixed task-level sequence, so we are able to form a complete task-motion plan p by simply chaining the sampled xy positions. Two consecutive pairs of xy positions (i.e., $\langle y_r^i[x], y_r^i[y] \rangle$ and $\langle y_r^{i+1}[x], y_r^{i+1}[y] \rangle$) can be used to parameterize a navigation action, and every single pair of xy positions plus an object position (i.e., y_o) can be used to parameterize a manipulation action. This enables us to convert a sample to a sequence of actions and then evaluate the sample by computing $\sum R(a)$. $\sum R(a)$ is the utility of a task-motion plan and serves as the objective function for the CMA-ES sampler.

5.6 Experiments

We conducted extensive experiments in simulation, where a mobile manipulator performs navigation and manipulation actions to "collect dishes" in a "restaurant" scenario. We also demonstrated the computed plan using our method on a real robot. Our main hy-

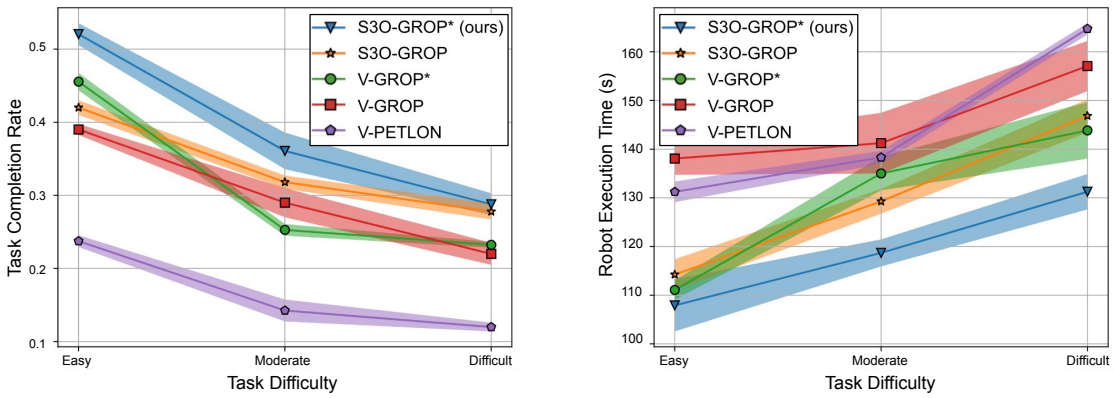


Figure 5.5: Overall performances of our approach (S3O-GROP*) and four baseline methods in task completion rate and robot execution time (s). Tasks are grouped based on their difficulties. S3O-GROP* produced the highest task completion rate while maintaining the lowest robot execution time. This observation is consistent over tasks of different difficulties.

pothesis is that under a planning time budget, the proposed framework outperforms existing TAMP algorithms in task completion rate and robot execution time.

5.6.1 Baselines

Ours and the baseline methods differ from each other in how to construct and optimize task planners (state spaces in particular). We compare S3O with basic object-centric Voronoi Partitioning (denoted as "V"). After selecting a task planner, there are different TAMP strategies we can choose from. We consider two TAMP algorithms for navigation domains from the literature, which are GROP [4] and PETLON [103]. Our TAMP component is built on GROP and further incorporates the proposed CMA-ES sampling algorithm for motion-level optimization. Thus, we denote our TAMP strategy as GROP*. Combining different methods from task planner construction and TAMP strategy, we consider the following five methods in total: S3O-GROP*, S3O-GROP, V-GROP*, V-GROP, and V-PETLON. We briefly summarize the major differences between the five methods:

- **S3O-GROP*** (proposed): It optimizes state spaces using S3O and samples navigation goals using CMA-ES. The algorithm optimizes efficiency and feasibility.
- **S3O-GROP**: An ablative version of S3O-GROP*. It is the same as S3O-GROP* without CMA-ES.
- **V-GROP***: An ablative version of S3O-GROP*. It is the same as S3O-GROP* without state space optimization.
- **V-GROP** [4]: It does not optimize the state space, and samples navigation goals only by feasibility. The algorithm optimizes plan efficiency and action feasibility.
- **V-PETLON** [103]: It does not optimize the state space, and selects navigation goals by just randomly sampling an obstacle-free position that is close to the object position. The algorithm optimizes plan efficiency but does not evaluate action feasibility.

In comparison, our method, S3O-GROP*, constructs the task planner using S3O and selects navigation goals by CMA-ES sampling, whose objective includes both motion-level feasibility and long horizon mobile manipulation cost. Note that we did not include "S3O-PETLON" as one of the baselines as there is no feasibility evaluation in the original PETLON algorithm, thus S3O is inapplicable.

5.6.2 Experimental Setup

The simulation environment contains seven tables of different sizes: one long table as the "bar area", two mid-sized tables, and four small tables that are able to take one person per table. Objects to be collected are randomly generated on the tables, and an obstacle

(i.e., chair) that is not mapped beforehand is placed near each object with a randomly generated position and orientation. The number of objects is dynamically changed for different environments, ranging from 5 to 7. An RGB camera is attached to the ceiling to capture overhead images of environments for robot perception. We assume the robot can hold multiple objects at the same time. Task completion is evaluated based on if "dishes" on the tables are successfully "collected".

The mobile manipulator in simulation includes a UR5e robot arm, a Robotiq 2F-140 gripper, an RMP 110 mobile base, and a Velodyne VLP-16 lidar sensor on the mobile base. We used the Building-Wide Intelligence (BWI) codebase [27] to construct our simulation platform, which relies on the Gazebo physics engine [29]. Rapidly exploring Random Tree (RRT) approach [24] is used to compute motion-level manipulation plans. The navigation stack was built using the `move_base` package of Robot Operating System (ROS) [26]. The robot's task planner is ASP-based [22, 23] and the Clingo solver is applied for computing task plans [122]. We adopted the FCN-VGG16 model [12] for predicting action feasibility heatmaps. The model is trained using a machine equipped with an Intel 3.80GHz i7-10700k CPU and a GeForce RTX 3070 GPU on a Ubuntu system.

5.6.3 Planning Parameters

At planning time, we do parallel computing using 12 CPUs on a different machine from training the FCN model. The machine for planning is equipped with an 11th Gen Intel(R) 2.30GHz Core(TM) i7-11800H CPU. The planning time budget is set to 300 seconds. For each task-level sequence, the maximum number of motion-level samples that can be drawn is 200. The manipulation constant cost δ is set to 5, and the navigation constant starting

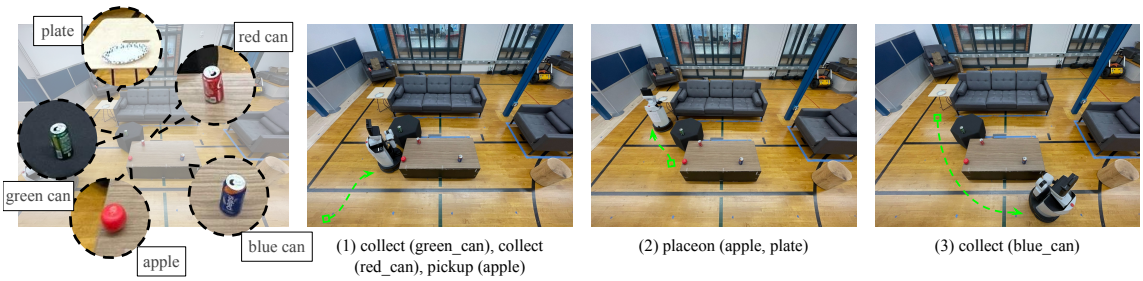


Figure 5.6: Real robot demonstration of the planned trajectory computed using our optimized task planner.

cost γ is set to 20. The reward for a successful manipulation action λ has a value of 150. The robot velocity v is set to 0.4m/s. For the CMA-ES sampler, we consider the first 20 generations. Since the number of motion-level samples is fixed (i.e., 200), the population size of each generation is set to 10. After ranking all possible state spaces, we choose the top 5 of them according to the computed scores.

5.6.4 Task Completion Rate and Robot Execution Time

Fig. 5.5 shows the main results of task completion rate and robot execution time. There were a total of 100 different tasks. We grouped the tasks based on their difficulties: Easy, Moderate, and Difficult. A task's *difficulty* is measured by the total area that a robot can navigate to and pick up an object from. For instance, a task with all feasible picking up positions being surrounded by obstacles has a high difficulty. After sorting the tasks based on their difficulties, we evenly placed them into the three groups.

Our system consistently performed the best in task completion rate (left subfigure) in all three settings, while maintaining the lowest robot execution time (right subfigure). We also see that methods that use S3O (i.e., S3O-GROP* and S3O-GROP) have better or at least

Table 5.1: Ablation study on the impact of different strategies for constructing the task planner. **Task completion rate / robot execution time** are reported in the table. S3O is our method that does task planner optimization; S3O-Random is an ablative version that uniformly selects the task planner from the candidate set.

Task Difficulty	S3O	S3O-Random
Easy	**0.52** $\pm$ 0.02 / 107.90 $\pm$ 5.35	0.30 $\pm$ 0.06 / **96.95** $\pm$ 2.40
Moderate	**0.36** $\pm$ 0.03 / 118.68 $\pm$ 2.78	0.18 $\pm$ 0.08 / **97.89** $\pm$ 4.22
Difficult	**0.29** $\pm$ 0.02 / 131.25 $\pm$ 3.64	0.17 $\pm$ 0.08 / **102.85** $\pm$ 6.30

similar performance compared with the methods that use basic Voronoi Partitioning (i.e., V-GROP*, V-GROP, and V-PETLON). While only considering methods that use Voronoi Partitioning, the one that uses GROP* generates plans that are of the least execution time and maintains a similar (higher) success rate as compared to V-GROP (V-PETLON). Both GROP* and GROP consider feasibility when sampling navigation points, but the former also takes efficiency into account by using CMA-ES. That is the reason why V-GROP* and V-GROP share similar success rates but the former performs better in plan efficiency. Overall, the results support our hypothesis.

5.6.5 Ablation Study

We also conducted an ablation study (as shown in TABLE 5.1) to learn the impact of different strategies for constructing the task planner, specifically how to select a state space from a set of state space candidates at planning time. Without a predefined state space, we compare two methods for state space selection: the proposed S3O (with score ranking), and an approach that uniformly samples state spaces from all possible candidates (denoted as "Random"). We observe that by considering S3O, the robot achieves a higher task completion rate for all tasks. When uniformly selects a state space to construct the task planner, the system produces more cost-efficient plans, however, suffers from very

poor performance in completing the task. The reason is that the random selection strategy treats every state space candidate equally, even though some state spaces are unreasonable for the current task. For instance, if two objects are too far from each other for the robot to reach both, it will be reasonable to separate the two objects into different locations instead of merging them into a single one. However, given the limited planning time, it is almost impossible for S3O-Random to select the most suitable state space especially when there are many objects, thus resulting in much lower task completion rates. On the other hand, it is expected to see more Voronoi area merging operations (including feasible and infeasible ones) for S3O-Random than our method which prefers only the feasible ones. As a result, the S3O-Random agent prefers to navigate only a few times and tries to complete the whole task, which is not ideal. In comparison, S3O (ours) seeks balance in task completion rate and robot execution time.

5.6.6 Real Robot Demonstration

We demonstrated the generated plan using S3O-GROP* on a real robot, as shown in Fig. 5.6. We use the Human Support Robot (HSR) from Toyota [144]. The robot is given a "tidy home" task, including collecting three empty cans and moving the apple to the white plate. Using our planning framework, the robot planned to navigate to the first position to do three manipulation actions: "collect" (i.e., pick up the object and put it into a garbage bag mounted on the robot) the green and red cans, and pick up the apple. While holding the apple in hand, the robot then went to the second position to place the apple on the plate. Finally, the robot planned to go to the third position to collect the blue can.

5.7 Conclusion

This chapter introduces Symbolic State Space Optimization (S3O), which constructs state space candidates from object-centric partitioning of the configuration space and ranks each candidate by probabilistically evaluating action feasibility values. S3O is applied to a TAMP system for long horizon mobile manipulation tasks where we further improve motion-level search efficiency using CMA-ES. The resulting framework is called S3O-GROP*, which was extensively evaluated in simulation and demonstrated it in real. Results showed that S3O-GROP* produces task-motion plans that are of higher quality than existing TAMP algorithms in terms of task completion rate and robot execution time.

6 Conclusions and Future Work

This dissertation thoroughly studies the problem of symbol grounding and grounded planning in the context of robotics.

For symbol grounding, we investigates the problem of Multimodal Embodied Attribute Learning (MEAL) that both require a robot to compute a policy of leveraging multimodal exploratory behaviors to identify object attributes. Accordingly, we have developed two algorithms called mixed observability robot control (MORC) and MORC with information-theoretic reward shaping (MORC-ITRS) for addressing OFFLINE- and ONLINE-MEAL problems respectively. MORC uses mixed observability Markov decision processes (MOMDPs) to solve OFFLINE-MEAL problems, where a robot selects actions for multimodal perception in object exploration tasks. Experimental results show that MORC enables the robot to identify object attributes more accurately without introducing extra cost from exploratory behaviors compared to a baseline that suggests actions following a predefined action sequence. We further present MORC-ITRS selects exploratory behaviors toward simultaneous attribute classification and attribute identification. The latter is built on MORC, and provoides an information-theoretic reward function for the exploration-exploitation trade off in ONLINE-MEAL problems. Experimental results show that MORC-ITRS enables the robot to complete attribute identification tasks at a higher accuracy using the same amount

of training time compared to baselines.

Then we apply grounded symbols to robot task and motion planning. Specifically, we discussed **G**rounded **RO**bot Task and Motion **P**lanning (GROP), that considers both efficiency and feasibility for robot task-motion planning. GROP visually grounds spatial relationships to probabilistically evaluate action feasibility, and is particularly suitable for TAMP domains with long-term robot operations (e.g., long-distance navigation). We extensively evaluated GROP in simulation using a mobile manipulator, and demonstrated it using a real robot system that includes a mobile robot and an arm robot. Results showed that GROP outperformed competitive baselines from the literature in plan efficiency without introducing additional action costs. As a follow-up research, **S**ymbolic **S**tate **S**pace **O**ptimization (S3O) was introduced. S3O constructs state space candidates from object-centric partitioning of the configuration space and ranks each candidate by probabilistically evaluating action feasibility values. We further improve motion-level search efficiency in S3O using CMA-ES. Results showed that the proposed method produces task-motion plans that are of higher quality than existing TAMP algorithms (including GROP) in terms of task completion rate and robot execution time.

6.1 Future Work I: Towards Transferable and Generalized Attribute Grounding in Multi-Robot Setting

One common limitation of the two algorithms we used for symbol grounding in MEAL problems is that the attribute classifiers are learned by a single robot and cannot directly be used by another robot that has different behaviors, morphology, and sensory modalities. It may be possible to use sensorimotor transfer learning (e.g., [145, 146, 147]) in future

work to scale up our framework to allow multiple different robots to learn such models and share their knowledge to further speed up learning. In addition, considering correlations between attributes and handling fuzzy attributes can potentially improve the performance of ONLINE-MEAL. Handling unseen attributes could be another interesting focus. Another direction for the future is to learn the world dynamics through the task completion process (currently the transition function is provided and the observation function is learned), where reinforcement learning methods potentially can be used. It is also important to consider human-robot dialogue to acquire attribute labels for objects in MEAL problems.

Multimodal perception is emphasized in MEAL agents, yet its challenges have been largely overlooked by robotic researchers. The heterogeneous nature of multimodal data makes the use of hand-designed features and multimodal sensor fusion extremely challenging for downstream robotic applications [8]. Additionally, collecting large-scale transferable datasets across different robot platforms requires substantial effort and is expensive, making multimodal sensor fusion and representation learning difficult for most robotics research labs to initiate. Recently, some researchers have started developing realistic multimodal simulations for robots, such as [148] for simulating audio data. However, these platforms are still experimental and in early stages. Therefore, we aim to explore methods for collecting large-scale multimodal data. both on real robots and in simulations.

To construct a more generalized framework for attribute grounding, we would like to explore the possibility of formulating MEAL problems especially ONLINE-MEAL as Bayes-Adaptive POMDPs (BAPOMDP) [149] where observation probabilities (functions) can be considered as unobserved parameters in the state space over which we maintain beliefs. During the learning process, the robot will continually gather data for approximating the

BAPOMDP model while maintaining attribute identification performance guarantees. Fundamentally, a BAPOMDP model enables sequencing actions with the optimal trade-off between exploration and exploitation, which is exactly the underlying challenge of ONLINE-MEAL. There can be practical challenges in applying BAPOMDP to our ONLINE-MEAL problems, such as the lack of systems for learning BAPOMDP policies and the necessity for more training data. BAPOMDP assumes the reward function is known and stationary. In our current formulation of ONLINE-MEAL, however, the reward function dynamically changes based on the learned observation function. Thus, a principled solution would require a new version of POMDP (and corresponding algorithms and systems), where both reward and observation functions are learned over time. Developing such a general-purpose framework would add a strong theoretical contribution to the literature, though in this article we chose to develop solutions (MORC and MORC-ITRS) focusing on the specific MEAL problems. There can be interesting future research to answer those questions.

6.2 Future Work II: Grounding Task Planners using Vision-Language Models

Classical planning methods that are used in TAMP are good at leveraging rule-based human knowledge to compute correct plans but suffer from the strong assumptions of perfect perception and action executions. For example, if the world model includes an apple on a table, classical planners assume that the robot will always locate the apple after reaching the table's location, and picking up the apple will deterministically result in it being in the robot's hand. These assumptions fail to consider dynamically changing environments and uncertain action outcomes, rendering it impractical for the robot to complete tasks by

simply following computed plans in the real world. To enable successful plan executions, TAMP frameworks are frequently accompanied by a plan monitoring system for linking the symbolic states and actions to robot sensory observations, where significant engineering efforts are needed. Given the natural connection between planning symbols and human language, it will be an interesting future direction to investigate how pre-trained Vision-Language Models (VLMs) can assist the robot in realizing symbolic plans generated by classical planners, while avoiding the engineering efforts of checking the outcomes of each action.

One way to do so is by leveraging VLMs to detect action failures and verify action affordances towards successful plan execution [150]. We can potentially take the advantage of the domain knowledge encoded in classical planners, including the actions defined by their effects and preconditions. For instance, by simply querying current observations against the action knowledge, similar to applying VLMs to Visual Question Answering (VQA) tasks, the robot can repeat an unsuccessful action or call the symbolic planner to generate a new valid plan. However, we also acknowledge that such VLMs are not meant to be trained for robot planning tasks, thus there will be more discussions on if researchers should spend time on designing interfaces to make good use of large models [19], or collect hardware and simulation data on training foundation models for robotics [151].

Another significant challenge in the field of planning and grounding is the unclear definition of grounded planning. Although there has been an increasing amount of research focusing on this topic, the lack of consistent assumptions or unified terminology within publicly released benchmarks makes it difficult for researchers to reproduce state-of-the-art results and collaborate effectively. For example, in perception, some work uses first-person

point clouds as observations, while others use bird-eye views; in planning, much research assumes the existence of pre-trained robot skills, but these skills depend on different robot platforms and vary significantly in task performance. To address these issues, we aim to benchmark our grounded TAMP research (i.e., GROP in Chapter 4 and S3O in Chapter 5) to facilitate progress in both the AI and robotics communities.

Regardless, we remain hopeful about successfully addressing these challenges in our future endeavors.

Bibliography

[1] Caelan Reed Garrett et al. “Integrated task and motion planning”. In: *Annual review of control, robotics, and autonomous systems* 4 (2021), pp. 265–293.

[2] Constructions Aeronautiques et al. “PDDL— The Planning Domain Definition Language”. In: *Technical Report, Tech. Rep.* (1998).

[3] Xiaohan Zhang et al. “Multimodal embodied attribute learning by robots for object-centric action policies”. In: *Autonomous Robots* (2023), pp. 1–24.

[4] Xiaohan Zhang et al. “Visually grounded task and motion planning for mobile manipulation”. In: *arXiv preprint arXiv:2202.10667* (2022), pp. 1925–1931.

[5] Xiaohan Zhang et al. “Symbolic State Space Optimization for Long Horizon Mobile Manipulation Planning”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 866–872.

[6] Stevan Harnad. “The symbol grounding problem”. In: *Physica D: Nonlinear Phenomena* 42.1-3 (1990), pp. 335–346.

[7] Saeid Amiri et al. “Multi-modal Predicate Identification using Dynamically Learned Robot Controllers.” In: *IJCAI*. 2018, pp. 4638–4645.

[8] Michelle A Lee et al. “Making sense of vision and touch: Learning multimodal representations for contact-rich tasks”. In: *IEEE Transactions on Robotics* (2020).

[9] Yoshio Yamamoto and Xiaoping Yun. “Coordinating locomotion and manipulation of a mobile manipulator”. In: *Proceedings of the 31st IEEE Conference on Decision and Control*. IEEE. 1992, pp. 2643–2648.

[10] Shantanu Thakar et al. “A Survey of Wheeled Mobile Manipulation: A Decision-Making Perspective”. In: *Journal of Mechanisms and Robotics* (2023).

[11] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Integrated task and motion planning in belief space”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[13] Kishan Chandan et al. “Learning to guide human attention on mobile telepresence robots with 360 vision”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5297–5304.

[14] Thomas Lew et al. “Robotic table wiping via reinforcement learning and whole-body trajectory optimization”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 7184–7190.

[15] Priyam Parashar et al. “SLAP: Spatial-Language Attention Policies”. In: *7th Annual Conference on Robot Learning*. 2023.

[16] Yan Ding et al. “Task-motion planning for safe and efficient urban driving”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[17] Yan Ding et al. “Learning to Ground Objects for Robot Task andMotion Planning”. In: *IEEE Robotics and Automation Letters (RA-L)*. Vol. 7. 2. IEEE, 2022, pp. 5536–5543.

[18] Bo Liu et al. “LLM+ P: Empowering Large Language Models with Optimal Planning Proficiency”. In: *arXiv preprint arXiv:2304.11477* (2023).

[19] Yan Ding et al. “Integrating Action Knowledge and LLMs for Task Planning and Situation Handling in Open Worlds”. In: *arXiv preprint arXiv:2305.17590* (2023).

[20] Yan Ding et al. “Task and motion planning with large language models for object rearrangement”. In: *arXiv preprint arXiv:2303.06247* (2023).

[21] Arjun Majumdar et al. “OpenEQA: Embodied Question Answering in the Era of Foundation Models”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024.

[22] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.

[23] Vladimir Lifschitz. “Answer set programming and plan generation”. In: *Artificial Intelligence* 138.1-2 (2002), pp. 39–54.

[24] Steven M LaValle et al. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998).

[25] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[26] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[27] Piyush Khandelwal et al. “Bwibots: A platform for bridging the gap between ai and human–robot interaction research”. In: *The International Journal of Robotics Research* 36.5-7 (2017), pp. 635–659.

[28] Sachin Chitta, Ioan Sucan, and Steve Cousins. “Moveit![ros topics]”. In: *IEEE Robotics & Automation Magazine* 19.1 (2012), pp. 18–19.

[29] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2004.

[30] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. New Jersey: John Wiley & Sons, 2014.

[31] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[32] Sylvie CW Ong et al. "Planning under uncertainty for robotic tasks with mixed observability". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1053–1068.

[33] Jesse Thomason et al. "Learning Multi-Modal Grounded Linguistic Semantics by Playing "I Spy"." In: *IJCAI*. 2016, pp. 3477–3483.

[34] Jivko Sinapov, Connor Schenck, and Alexander Stoytchev. "Learning relational object categories using behavioral exploration and multimodal perception". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 5691–5698.

[35] Gyan Tatiya and Jivko Sinapov. "Deep multi-sensory object category recognition using interactive behavioral exploration". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7872–7878.

[36] Jesse Thomason et al. "Opportunistic active learning for grounding natural language descriptions". In: *Conference on Robot Learning*. PMLR. 2017, pp. 67–76.

[37] Jesse Thomason et al. "Guiding exploratory behaviors for multi-modal grounding of linguistic descriptions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[38] Jivko Sinapov and Alexander Stoytchev. "Grounded object individuation by a humanoid robot". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 4981–4988.

[39] Jivko Sinapov et al. "Grounding semantic categories in behavioral interactions: Experiments with 100 objects". In: *Robotics and Autonomous Systems* 62.5 (2014), pp. 632–645.

[40] Xiaohui Chen et al. "A Framework for Multisensory Foresight for Embodied Agents". In: *2021 IEEE International Conference on Robotics and Automation*. IEEE. 2021.

[41] Xiaohan Zhang, Jivko Sinapov, and Shiqi Zhang. "Planning multimodal exploratory actions for online robot attribute learning". In: *arXiv preprint arXiv:2106.03029* (2021).

[42] Olga Russakovsky and Li Fei-Fei. "Attribute learning in large-scale datasets". In: *European Conference on Computer Vision*. Springer. 2010, pp. 1–14.

[43] Steven Chen and Kristen Grauman. "Compare and contrast: Learning prominent visual differences". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1267–1276.

[44] Vittorio Ferrari and Andrew Zisserman. "Learning visual attributes". In: *Advances in neural information processing systems* 20 (2007), pp. 433–440.

[45] Ali Farhadi et al. “Describing objects by their attributes”. In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 1778–1785.

[46] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. “Learning to detect unseen object classes by between-class attribute transfer”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 951–958.

[47] Dinesh Jayaraman and Kristen Grauman. “Zero shot recognition with unreliable attributes”. In: *arXiv preprint arXiv:1409.4327* (2014).

[48] Ziad Al-Halah, Makarand Tapaswi, and Rainer Stiefelhagen. “Recovering the missing link: Predicting class-attribute associations for unsupervised zero-shot learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5975–5984.

[49] Mengye Ren et al. “Flexible Few-Shot Learning with Contextual Similarity”. In: *arXiv preprint arXiv:2012.05895* (2020).

[50] Devi Parikh and Kristen Grauman. “Relative attributes”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 503–510.

[51] Genevieve Patterson and James Hays. “Coco attributes: Attributes for people, animals, and objects”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 85–100.

[52] Ranjay Krishna et al. “Visual genome: Connecting language and vision using crowdsourced dense image annotations”. In: *International journal of computer vision* 123.1 (2017), pp. 32–73.

[53] Khoi Pham et al. “Learning to Predict Visual Attributes in the Wild”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13018–13028.

[54] Dmitry Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).

[55] Yuke Zhu et al. “Target-driven visual navigation in indoor scenes using deep reinforcement learning”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3357–3364.

[56] Stefanie Tellex et al. “Robots that use language”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2020), pp. 25–55.

[57] Ravinder S Dahiya et al. “Tactile sensing—from humans to humanoids”. In: *IEEE transactions on robotics* 26.1 (2009), pp. 1–20.

[58] Qiang Li et al. “A review of tactile information: Perception and action through touch”. In: *IEEE Transactions on Robotics* 36.6 (2020), pp. 1619–1634.

[59] Javier Monroy et al. “A semantic-based gas source localization with a mobile robot combining vision and chemical sensing”. In: *Sensors* 18.12 (2018), p. 4174.

[60] Bianca Ciui et al. “Chemical sensing at the robot fingertips: Toward automated taste discrimination in food samples”. In: *ACS sensors* 3.11 (2018), pp. 2375–2384.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[62] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proc. of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[63] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[64] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[65] Eleanor J Gibson. "Exploratory behavior in the development of perceiving, acting, and the acquiring of knowledge". In: *Annual review of psychology* 39.1 (1988), pp. 1–42.

[66] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[67] Dermot Lynott and Louise Connell. "Modality exclusivity norms for 423 object properties". In: *Behavior Research Methods* 41.2 (2009), pp. 558–564.

[68] Jeannette Bohg et al. "Interactive perception: Leveraging action in perception and perception in action". In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1273–1291.

[69] Yang Gao et al. "Deep learning for tactile understanding from visual and haptic data". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 536–543.

[70] Matthias Kerzel et al. "Neuro-robotic haptic object classification by active exploration on a novel dataset". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[71] Dhiraj Gandhi, Abhinav Gupta, and Lerrel Pinto. "Swoosh! Rattle! Thump!–Actions that Sound". In: *arXiv preprint arXiv:2007.01851* (2020).

[72] Raphaël Braud et al. "Robot multi-modal object perception and recognition: synthetic maturation of sensorimotor learning in embodied systems". In: *IEEE Trans. on Cognitive and Developmental Systems* (2020).

[73] Jacob Arkin et al. "Multimodal estimation and communication of latent semantic knowledge for robust execution of robot instructions". In: *The International Journal of Robotics Research* 39.10-11 (2020), pp. 1279–1304.

[74] Michelle A Lee et al. "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8943–8950.

[75] Chen Wang et al. "SwingBot: Learning Physical Features from In-hand Tactile Exploration for Dynamic Swing-up Manipulation". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*. 2020, pp. 5633–5640.

[76] Jeremy A Fishel and Gerald E Loeb. “Bayesian exploration for intelligent identification of textures”. In: *Frontiers in neurorobotics* 6 (2012), p. 4.

[77] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Cambridge: MIT press, 2018.

[78] Robert Platt Jr et al. “Belief space planning assuming maximum likelihood observations”. In: (2010).

[79] Stéphane Ross et al. “A Bayesian Approach for Learning and Planning in Partially Observable Markov Decision Processes.” In: *J. of Machine Learning Research* 12.5 (2011).

[80] Mohan Sridharan, Jeremy Wyatt, and Richard Dearden. “Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs”. In: *Artificial Intelligence* 174.11 (2010), pp. 704–725.

[81] Robert Eidenberger and Josef Scharinger. “Active perception and scene modeling by planning with probabilistic 6d object poses”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 1036–1043.

[82] Kaiyu Zheng et al. “Multi-Resolution POMDP Planning for Multi-Object Search in 3D”. In: *arXiv preprint arXiv:2005.02878* (2020).

[83] Shiqi Zhang, Mohan Sridharan, and Christian Washington. “Active visual planning for mobile robot teams using hierarchical POMDPs”. In: *IEEE Transactions on Robotics* 29.4 (2013), pp. 975–985.

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

[85] Jivko Sinapov et al. “Learning to Order Objects Using Haptic and Proprioceptive Exploratory Behaviors.” In: *IJCAI*. 2016, pp. 3462–3468.

[86] Aitor Aldoma, Federico Tombari, and Markus Vincze. “Supervised learning of hidden and non-hidden 0-order affordances and detection in real scenes”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 1732–1739.

[87] Michael N Katehakis and Arthur F Veinott Jr. “The multi-armed bandit problem: decomposition and computation”. In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268.

[88] Shiqi Zhang, Piyush Khandelwal, and Peter Stone. “Dynamically constructed (PO) MDPs for adaptive robot planning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[89] Shiqi Zhang and Peter Stone. “iCORPP: Interleaved Commonsense Reasoning and Probabilistic Planning on Robots”. In: *arXiv preprint arXiv:2004.08672* (2020).

[90] Hanna Kurniawati, David Hsu, and Wee Sun Lee. “Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces.” In: *Robotics: Science and systems*. Vol. 2008. Citeseer. 2008.

[91] Fabien Lagriffoul et al. “Platform-independent benchmarks for task and motion planning”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3765–3772.

[92] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[93] Howie M Choset et al. *Principles of robot motion: theory, algorithms, and implementation*. MIT Press, 2005.

[94] Marc Toussaint. “Logic-geometric programming: an optimization-based approach to combined task and motion planning”. In: *The 24th International Conference on Artificial Intelligence*. 2015, pp. 1930–1936.

[95] Yifeng Zhu et al. “Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs”. In: *ICRA* (2020).

[96] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. “FFRob: Leveraging symbolic planning for efficient task and motion planning”. In: *The International Journal of Robotics Research* 37.1 (2018), pp. 104–136.

[97] Yu-qian Jiang et al. “Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems”. In: *Frontiers of Information Technology & Electronic Engineering* 20.3 (2019), pp. 363–373.

[98] Antony Thomas, Fulvio Mastrogiovanni, and Marco Baglietto. “MPTP: Motion-planning-aware task planning for navigation in belief space”. In: *Robotics and Autonomous Systems* 141 (2021), p. 103786.

[99] Andrew M Wells et al. “Learning feasibility for task and motion planning in tabletop environments”. In: *IEEE robotics and automation letters* 4.2 (2019), pp. 1255–1262.

[100] Toki Migimatsu and Jeannette Bohg. “Object-centric task and motion planning in dynamic environments”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 844–851.

[101] James McMahon and Erion Plaku. “Robot motion planning with task specifications via regular languages”. In: *Robotica* 35.1 (2017), pp. 26–49.

[102] Ye Zhao et al. “Reactive task and motion planning for robust whole-body dynamic locomotion in constrained environments”. In: *arXiv preprint arXiv:1811.04333* (2018).

[103] Shih-Yun Lo, Shiqi Zhang, and Peter Stone. “The PETLON Algorithm to Plan Efficiently for Task-Level-Optimal Navigation”. In: *Journal of Artificial Intelligence Research* 69 (2020), pp. 471–500.

[104] Danny Driess et al. “Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 9563–9569.

[105] Fabien Gravot, Stephane Cambon, and Rachid Alami. “aSyMov: a planner that deals with intricate symbolic and geometric problems”. In: *Robotics Research. The Eleventh International Symposium*. Springer. 2005, pp. 100–110.

[106] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. "Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning." In: *Robotics: Science and Systems*. 2007, pp. 326–333.

[107] Esra Erdem et al. "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4575–4581.

[108] Siddharth Srivastava et al. "Combined task and motion planning through an extensible planner-independent interface layer". In: *IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 639–646.

[109] Fabien Lagriffoul et al. "Efficiently combining task and motion planning using geometric constraints". In: *The International Journal of Robotics Research* 33.14 (2014), pp. 1726–1747.

[110] Rohan Chitnis et al. "Guided search for task and motion plans using learned heuristics". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 447–454.

[111] Zi Wang et al. "Active model learning and diverse action sampling for task and motion planning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4107–4114.

[112] Beomjoon Kim et al. "Learning to guide task and motion planning using score-space representation". In: *The International Journal of Robotics Research* 38.7 (2019), pp. 793–812.

[113] Rohan Chitnis, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Learning quickly to plan quickly using modular meta-learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7865–7871.

[114] Beomjoon Kim and Luke Shimanuki. "Learning value functions with relational state representations for guiding task-and-motion planning". In: *Conference on Robot Learning*. PMLR. 2020, pp. 955–968.

[115] Neil T Dantam et al. "An incremental constraint-based framework for task and motion planning". In: *The International Journal of Robotics Research* 37.10 (2018), pp. 1134–1151.

[116] Dylan Hadfield-Menell et al. "Modular task and motion planning in belief space". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015.

[117] Camille Phiquepal and Marc Toussaint. "Combined task and motion planning under partial observability: An optimization-based approach". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[118] Caelan Reed Garrett et al. "Online replanning in belief space for partially observable task and motion problems". In: *IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 5678–5684.

[119] Ahmed Nouman, Volkan Patoglu, and Esra Erdem. “Hybrid conditional planning for robotic applications”. In: *The International Journal of Robotics Research* 40.2-3 (2021), pp. 594–623.

[120] Aliakbar Akbari, Mohammed Diab, and Jan Rosell. “Contingent task and motion planning under uncertainty for human–robot interactions”. In: *Applied Sciences* 10.5 (2020), p. 1665.

[121] 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*. 2020.

[122] Martin Gebser et al. “Clingo= ASP+ control: Preliminary report”. In: *arXiv preprint arXiv:1405.3694* (2014).

[123] Atsuyuki Okabe and Atsuo Suzuki. “Locational optimization problems solved through Voronoi diagrams”. In: *European journal of operational research* (1997).

[124] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)”. In: *Evolutionary computation* (2003).

[125] Yan Ding et al. “GLAD: Grounded Layered Autonomous Driving for Complex Service Tasks”. In: *arXiv preprint arXiv:2210.02302* (2022).

[126] Zi Wang et al. “Learning compositional models of robot skills for task and motion planning”. In: *The International Journal of Robotics Research* 40.6-7 (2021), pp. 866–894.

[127] Nakul Gopalan et al. “Simultaneously learning transferable symbols and language groundings from perceptual data for instruction following”. In: *Robotics: Science and Systems XVI* (2020).

[128] Naman Shah and Siddharth Srivastava. “Using deep learning to bootstrap abstractions for hierarchical robot planning”. In: *arXiv preprint arXiv:2202.00907* (2022).

[129] Dmitry Berenson, James Kuffner, and Howie Choset. “An optimization approach to planning for mobile manipulation”. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1187–1192.

[130] Rosen Diankov and James Kuffner. “Openrave: A planning architecture for autonomous robotics”. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34* 79 (2008).

[131] Freek Stulp et al. “Learning and reasoning with action-related places for robust mobile manipulation”. In: *Journal of Artificial Intelligence Research* 43 (2012), pp. 1–42.

[132] Shunan Ren et al. “A method for optimizing the base position of mobile painting manipulators”. In: *IEEE Transactions on Automation Science and Engineering* 14.1 (2016), pp. 370–375.

[133] Franziska Zacharias et al. “Positioning mobile manipulators to perform constrained linear trajectories”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 2578–2584.

[134] Nikolaus Vahrenkamp, Tamim Asfour, and Rüdiger Dillmann. "Robot placement based on reachability inversion". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1970–1975.

[135] Snehal Jauhri, Jan Peters, and Georgia Chalvatzaki. "Robot learning of mobile manipulation with reachability behavior priors". In: *IEEE Robotics and Automation Letters* (2022).

[136] Jiayuan Gu et al. "Multi-skill Mobile Manipulation for Object Rearrangement". In: *arXiv preprint arXiv:2209.02778* (2022).

[137] Fei Xia et al. "Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation". In: *ICRA* (2021).

[138] Chengshu Li et al. "Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators". In: *Conference on Robot Learning*. 2020.

[139] Thomas Lew et al. "Robotic Table Wiping via Reinforcement Learning and Whole-body Trajectory Optimization". In: *ICRA* (2023).

[140] WF Carriker, PK Khosla, and BH Krogh. "Path planning for mobile manipulators for multiple task execution". In: *IEEE Transactions on Robotics and Automation* 7.3 (1991), pp. 403–408.

[141] Jingren Xu et al. "Planning an efficient and robust base sequence for a mobile manipulator performing multiple pick-and-place tasks". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 11018–11024.

[142] Fabian Reister, Markus Grotz, and Tamim Asfour. "Combining Navigation and Manipulation Costs for Time-Efficient Robot Placement in Mobile Manipulation Tasks". In: *IEEE Robotics and Automation Letters* (2022).

[143] Chen Wang et al. "Densefusion: 6d object pose estimation by iterative dense fusion". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

[144] Takashi Yamamoto et al. "Development of human support robot as the research platform of a domestic mobile manipulator". In: *ROBOMECH journal* (2019).

[145] G. Tatiya et al. "Haptic Knowledge Transfer Between Heterogeneous Robots using Kernel Manifold Alignment". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2020.

[146] Gyan Tatiya et al. "A Framework for Sensorimotor Cross-Perception and Cross-Behavior Knowledge Transfer for Object Categorization". In: *Frontiers in Robotics and AI* 7 (2020), p. 137.

[147] Gyan Tatiya, Jonathan Francis, and Jivko Sinapov. "Transferring Implicit Knowledge of Non-Visual Object Properties Across Heterogeneous Robot Morphologies". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023.

[148] Chuang Gan et al. "Threedworld: A platform for interactive multi-modal physical simulation". In: *arXiv preprint arXiv:2007.04954* (2020).

[149] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. “Bayes-adaptive pomdps”. In: *Advances in neural information processing systems* 20 (2007).

[150] Xiaohan Zhang et al. “Grounding Classical Task Planners via Vision-Language Models”. In: *arXiv preprint arXiv:2304.08587* (2023).

[151] Abhishek Padalkar et al. “Open x-embodiment: Robotic learning datasets and rt-x models”. In: *arXiv preprint arXiv:2310.08864* (2023).