TASK AND MOTION PLANNING FOR ROBOTS IN OPEN WORLDS

BY

YAN DING

BE, Chongqing University, 2016
MS, Chongqing University, 2019

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2024

Accepted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2024

February 17, 2024

Shiqi Zhang, Chair
Department of Computer Science, Binghamton University

Patrick Madden, Member
Department of Computer Science, Binghamton University

Zhongfei (Mark) Zhang, Member
Department of Computer Science, Binghamton University

Ye Zhao, Member
George Woodruff School of Mechanical Engineering, Georgia Tech

Tara Dhakal, Outside Examiner
Electrical and Computer Engineering, Binghamton University

Abstract

To complete long-horizon tasks, real-world robots generally need to plan at both task and motion levels. Task-level planning involves generating a sequence of symbolic actions in a discrete space for long-term goals. In contrast, motion-level planning implements these actions by computing trajectories in a continuous space. Real-world robots usually operate within the context of "open-world" environments, characterized by their unpredictable and dynamic nature. Existing task and motion planning frameworks, however, tend to focus on closed-world settings, which assume the robot is provided with complete world knowledge. This discrepancy poses several challenges. At times, robots may receive underspecified requests from users. In these instances, robots should understand the requests and generate plans that meet user preferences. Additionally, robots might encounter new objects during task execution. Adapting their plans in response to these new objects is crucial. Moreover, robots must manage uncertainties from other agents, such as humans or robots, in open-world settings. Despite having a plan, they may face unexpected situations that hinder task completion.

The primary contribution of this research is to develop task and motion planning algorithms and systems that are efficient, feasible, and adaptable for robots operating in open-world environments. This dissertation comprises several studies. I begin by introducing

TMPUD, a task and motion planning framework for urban autonomous driving under full observability. TMPUD is designed to handle the uncertainty from other agents, aiming to enhance the safety and efficiency of urban driving. In contrast, the second framework, GLAD, operates under partial observability. The third framework, TMOC, focuses on planning with novel objects. This framework incorporates a physics engine to ground objects, learn about their physical attributes, and improve task-motion planning skills based on gathered learning experiences. Next, I discuss LLM-GROP, a method that leverages commonsense knowledge for planning to handle underspecified user requests in tasks involving multi-object rearrangement. Finally, I present COWP, a system that combines task-oriented commonsense knowledge derived from Large Language Models (LLMs) with classical AI planning methods to effectively handle unforeseen situations.

Acknowledgements

I am delighted to have completed this doctoral dissertation. The journey through my Ph.D. was filled with happiness, challenges, confusion, and uncertainty. However, I firmly believe that hard work pays off. I hope to achieve greater accomplishments in the field of robotics in the future and look forward to seeing robots serve humanity, liberating human labor, and making the world a better place.

The completion of this dissertation could not have been possible without the invaluable support of many. I am deeply grateful for their assistance. First and foremost, I would like to express my heartfelt thanks to my supervisor, Prof. Shiqi Zhang, for his unwavering support and guidance throughout my doctoral studies. I am also extremely grateful to Prof. Patrick Madden, Prof. Zhongfei (Mark) Zhang, and Prof. Ye Zhao for agreeing to be part of my committee and providing insightful feedback. Additionally, my thanks to Professor Tara Dhakal for serving as the outside examiner.

I would like to thank my fellow researchers in our lab, including Xiaohan Zhang, Xingyue Zhan, Yohei Hayamizu, Kishan Chandan, and David Defazio. Our collaborative efforts and discussions have created a rich environment that was instrumental in my contributions.

Finally, I would like to extend my deepest gratitude to my parents and my partner, Nieqing Cao, a Ph.D. candidate at Binghamton University in Systems Science and Industrial Engineering. Their unwavering encouragement, moral support, and guidance have been indispensable. Their support has been a cornerstone throughout my journey.

Contents

List of Tables

List of Figures

List of Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

ASP Answer Set Programming

BERT Bidirectional Encoder Representations from Transformers

C-space Configuration space

CARLA Car Learning to Act

CLIP Contrastive Language-Image Pre-training

COWP Common sense-based Open-World Planning

CW Closed World

CWA Closed World Assumption

EK External Knowledge

GGCNN Generative Grasping Convolutional Neural Network

GLAD Grounded Layered Autonomous Driving

GPT Generative Pre-training Transformer

GQCNN Grasp Quality Convolutional Neural Network

HPN Hierarchical Planning in the Now

KRR Knowledge Representation and Reasoning

LATP LLM-based Arrangement and Task Planning

LLM Large Language Model

LLM-GROP Large Language Model for Grounded Robot Task and Motion Planning

LMZSP Language Models as Zero-Shot Planners

MDP Markov Decision Processes

NLLLOSS Negative Log-Likelihood Loss

NLP Natural Language Processing

No-com No-communication

PDDL Planning Domain Definition Language

PID Proportional-Integral-Derivative

POI Points of Interest

PPS Task and Motion Planning with Physics-based Simulation

PRM Probabilistic Roadmaps

ROS Robot Operating System

RRT Rapidly-exploring Random Trees

SVC Support Vector Classification

SVM Support-Vector machine

TAMP Task and Motion Planning

Th-based Threshold-based

TMOC Task-Motion Object-Centric planning

TMPUD Task-Motion Planning for Urban Driving

TPRA Task Planning with Random Arrangement

VLM Vision Language Model

1 Introduction

"Planning" is a crucial topic in robotics, enabling service robots to function autonomously and efficiently in a wide array of tasks, such as navigation and manipulation [1]. The demand for service robots to perform planning in open-worlds has risen significantly. Service robots like household assistants, food delivery robots, and medical service robots operate in environments such as homes, offices, hospitals, and restaurants. These "open worlds" are characterized by a wide variety of unpredictable situations, which requires robots to interpret and adapt to successfully accomplish the service requests from users [2, 3]. A classic definition of "situation" is the complete state of affairs at some instant of time [4]. Within the context of service robotics, "situation" refers to a world state with the potential of preventing a robot from completing its current task using a solution that normally works [5].

Classical planning methods, typically devised based on the "closed world" assumption, asserting that the knowledge base encompasses all true statements, and any unexpressed claim is deemed false or unknown [6], usually struggle to address the challenges of open-world environments [7]. Consider a dining domain containing tables, chairs, food, tableware, and people. This domain may include unforeseen situations, such as broken cups or missing forks that can prevent a robot's task completion. Due to the complexity of "world

openness," it is impossible to foresee all potential situations. Additionally, real-world user instructions tend to be underspecified [8, 9, 10]. For instance, when a robot is tasked with setting a dinner table, there are many acceptable arrangements, but some are more preferable over others.

Robot planning becomes more difficult when further considering the complexity of tasks in open-world environments, which involves interactions with novel objects with unknown attributes (e.g., shapes and weights). To illustrate, consider the context of a dining environment. Questions such as, "Is there still room on the table to accommodate more objects?" and "What happens when objects interact physically, such as stacking plates and bowls?" add another layer of complexity to planning in open worlds.

This research discusses four key characteristics of open worlds within the context of robot planning: **uncertainty in the environment**, **novel objects**, **underspecified user requests**, and **unforeseen situations**. The objective of this prospectus is to develop planning systems that are not only efficient and feasible, but also flexible enough to adapt to unexpected challenges in open-world environments. Our approach integrates commonsense reasoning, which is generally applicable across many domains, with rule-based task planners, thus enabling the interpretation of complex world states. This combination leverages the complementary strengths of commonsense reasoning and automated planning to address the dynamic nature of open-world environments.

1.1 Research themes

My work focuses on five key aspects to address the challenges posed by open-world environments for task and motion planning frameworks. These challenges include uncertainty

in the environment, novel objects, underspecified requests, and unforeseen situations.

I) Planning under uncertainty with full observability: Autonomous driving technologies have the potential to transform urban mobility, provided they efficiently execute tasks while ensuring safety. To be considered effective, these vehicles need to compute a sequence of symbolic actions - such as merging left, going straight, turning left, parking right - to fulfill service requests while maintaining safety. Despite the widespread research on individual task planning and motion planning, there is a noticeable gap in literature focusing on the interaction between these two aspects. Hence, there is an urgent need for developing algorithms that bridge this gap, simultaneously improving task-completion efficiency and ensuring safe driving behaviors. *A precondition to this approach is full observability, where the autonomous vehicle has access to all relevant environmental and road user information.*

In Chapter 3, I introduce a novel algorithm called Task-Motion Planning for Urban Driving (TMPUD). This method marks the first time that interaction between task and motion planners has been achieved, integrating motion-level safety estimation and task-level replanning capabilities. It enables a new safety estimator and the TMPUD algorithm, considering uncertainties from both the vehicle itself and its surroundings. By quantitatively evaluating these uncertainties at the motion level, they are accounted for in the task planning, improving the overall efficacy and safety of autonomous driving. For example, if the left lane is busy and missing the next crossing does not add significant extra distance, the task planner can avoid instructing the vehicle to merge left.

TMPUD was implemented and evaluated using CARLA, an autonomous driving platform [11] that simulates urban driving scenarios. The results suggested that TMPUD out-

performs two baseline methods from existing literature in terms of both safety and efficiency. It shows promising potential for real-world applications, as it provides an effective bridge between task planning and motion planning, thus making autonomous urban driving safer and more efficient. I present the details of this work in Chapter 3, titled Planning with Uncertainty Under Full Observability.

II) Planning under uncertainty with partial observability: In Chapter 3, the TMPUD framework operates under the assumption of full observability, wherein the trajectories of all vehicles, including the ego vehicle and others, within a future time frame are known and accessible. However, this assumption often proves impractical due to the usual scenario of partial observability, where the ego vehicle has limited access to information.

To address this limitation, Chapter 4 introduces a novel algorithm, Grounded Layered Autonomous Driving (GLAD). This framework facilitates complex service requests in autonomous urban driving under conditions of partial observability. There are three layers for service-level, behavior-level, and motion-level decision-making. The layered framework is unique in its tight coupling, where the different layers communicate user preferences, safety estimates, and motion costs for system optimization. GLAD is visually grounded by perceptual learning from a dataset of $13.8k$ instances collected from driving behaviors. GLAD enables autonomous vehicles to efficiently and safely fulfill complex service requests. Experimental results from abstract and full simulation show that our system outperforms a few competitive baselines from the literature. The complete details of this work are presented in Chapter 4, titled Planning with Uncertainty Under Partial Observability.

We used the CARLA simulator [11] to collect a dataset of $13.8k$ instances, each in-

cluding 16 images, for evaluating the safety levels of driving behaviors. Learning from our gathered dataset enables GLAD to visually ground symbolic predicates for planning driving behaviors. We have compared GLAD with baseline methods [12, 13] that support decision-making at behavioral and motion levels. Results show that GLAD produced the highest overall utility compared to the baseline methods.

III) Planning with novel objects: Planning algorithms, whether they operate at the task, motion, or both levels, frequently assume the existence of a world model that includes the dynamics of how the world reacts to robot actions. Modeling all objects and their physical properties during planning is often impractical due to the immense complexity of real-world scenarios. For example, in environments like kitchens or shopping centers, the variety and interaction of objects increase the challenge. Simulating a robot's interaction with multiple objects also is extremely difficult. For illustration, consider a complex multi-object interaction task called "stack-and-push". The robot is tasked with constructing a tower of blocks and moving it to a predetermined area. This process requires the robot to learn properties of objects (like size and weight), to understand the grasping mechanism, which relies on these object properties, and to devise a strategy for stacking and pushing (such as deciding the number of blocks to stack).

In Chapter 5, I focus on the development and evaluation of Task-Motion Object-Centric planning (TMOC). This grounded task and motion planning (TAMP) algorithm is designed for complex real-world scenarios. I introduce Task and Motion Planning with Physics-based Simulation (PPS), a new grounded, object-centric TAMP framework. It incorporates a physics engine that grounds objects and their physical properties. This inclusion enables

the robot to simulate interactions and enhance policy learning. TMOC is designed to address PPS problems where object properties are not provided. It uses a high-fidelity physics engine to ground objects, learn object properties, and improve task-motion planning skills.

TMOC, evaluated in both simulation and on a real robot, outperformed competitive baselines in terms of cumulative utility, showcasing its capability to handle complex multi-object dynamic interactions. By enabling robots to better understand and interact with objects in complex environments, this work has the potential to significantly improve the autonomy and effectiveness of robots in real-world tasks. I present the details of this work in Chapter 5, titled Grounded Planning with Novel Objects.

IV) Planning to fulfill underspecified requests: Existing mobile manipulation systems often require explicit instructions to arrange objects, which are not readily available in real-world situations due to the frequent underspecification of user instructions. This poses a challenge, as tasks such as correctly setting a table necessitate considerable commonsense knowledge. The machine learning methods available today attempt to provide this knowledge, but their applicability to complex service tasks is limited by the requirement of extensive training data.

Chapter 6 presents an exploration into multi-object rearrangement tasks by service robots, a critical skill needed for tasks such as setting tables, organizing bookshelves, and loading dishwashers. It introduces an approach called LLM-GROP (Large Language Model for Grounded Robot Task and Motion Planning) that leverages commonsense knowledge for planning to complete object rearrangement tasks.

LLM-GROP, a method developed by our team, utilizes large language models (LLMs)

to generate symbolic and geometric spatial relationships between objects. The generated relationships are then evaluated for feasibility by a motion planning system. Finally, the system optimizes the feasibility and efficiency of different task-motion plans towards maximizing long-term utility. LLM-GROP was put into practice in a dining domain, where a mobile manipulator is tasked with setting a table as per user instructions. The robot, given a set of tableware, must calculate a tabletop configuration that aligns with common sense and then devise a task-motion plan to actualize the configuration.

LLM-GROP was evaluated by having users rate different place settings. The results showed that it improved user satisfaction compared to existing object rearrangement methods, while maintaining similar or lower cumulative action costs. LLM-GROP was also successfully demonstrated on a real robot. I present the details of this work in Chapter 6, titled Planning for Underspecified Requests.

V) Planning with Large Language Models to handle unforeseen situations: Most existing task planning systems for robots are designed for closed-world scenarios, assuming the robot is equipped with complete world knowledge. However, in open-world contexts where unforeseen situations are a common occurrence, such an assumption can potentially break the planner's completeness. Therefore, the challenge lies in making task planning systems robust in dealing with unexpected situations in an open-world context.

In Chapter 7, I introduce a novel system, called Common sense-based Open-World Planning (COWP). This system aims to improve robot task planning and handling in open-world scenarios. The COWP framework leverages Large Language Models (LLMs) [14] to dynamically augment a robot's action knowledge with task-oriented common sense. It

combines classical AI planning methods and LLM-based knowledge extraction to adapt to novel situations.

In operation, the system works by creating an initial plan based on user-provided goals, sequentially performing actions, evaluating the feasibility of the current plan, and then adapting it based on acquired commonsense knowledge. Systematic evaluations were performed using 12 dining-focused tasks from the ActivityPrograms dataset [15] and a dataset of over a thousand situations gathered from crowdsourced participants. The COWP approach outperformed competitive baselines from the literature in terms of the success rate of task completion. I present the details of this work in Chapter 7, titled Planning with Unforeseen Situations.

1.2 Dissertation Organization

In Chapter 2, I present the foundational information regarding grounded planning in open worlds for service robots. Next, I detail my contributions in Chapters 3 through 7. These chapters align with my published or accepted papers [13, 16, 17, 10, 5], respectively. The dissertation concludes with a summary and an outlook on future work in Chapter 8.

2 Background

In this section, important ideas related to this study are discussed. Automated planning is introduced first. Then, an explanation of what motion planning is, is provided. This is followed by a look at object grasping. Afterwards, large language models and vision language models are explored. Lastly, a mobile manipulator, used in this research, is discussed. It is expected that this section will aid readers in better understanding the subsequent parts of the study.

2.1 Task 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 [18]. PDDL aims to standardize the formu-

lation 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 [19,

20]. 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) [21] and the Probabilistic Roadmaps (PRM) [22].

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 Object Grasping

Object grasping is a fundamental capability required by service robots to physically interact with their environment. The ability to grasp and manipulate objects allows robots to perform tasks in a variety of real-world scenarios, from household chores to industrial automation. The development of effective grasping algorithms has been a significant challenge due to the complexity of the physical world and the uncertainty associated with the

sensory information. Among the various solutions, two approaches stand out, namely, Generative Grasping Convolutional Neural Network (GGCNN) [23] and Grasp Quality Convolutional Neural Network (GQCNN) [24].

GGCNN: The Generative Grasping Convolutional Neural Network (GGCNN) is a deep learning-based method developed for real-time, pixel-wise prediction of grasp quality and pose directly from a single depth image. The key advantage of GGCNN is its ability to generate a full grasp configuration without the need for sampling and ranking potential grasps, which makes it significantly faster and more efficient than traditional methods.

GGCNN is composed of an encoder-decoder architecture that takes a depth image as input and generates four output channels representing the grasp pose (x, y), grasp angle, and grasp width. The model's quality estimation ability comes from the simultaneous regression of these four output channels. This process enables GGCNN to propose an optimal grasp for the given object, which can then be executed by a robot manipulator.

GQCNN: The Grasp Quality Convolutional Neural Network (GQCNN) is another innovative method for object grasping. It also utilizes a depth image for grasp prediction but differs from GGCNN by incorporating a measure of grasp robustness or quality.

The GQCNN model takes as input a depth image patch centered at the grasp contact point and outputs a binary classification of whether the grasp will succeed or not. Additionally, GQCNN can provide a continuous score representing the quality of the predicted grasp. The inclusion of grasp quality in the decision process makes GQCNN highly effective in selecting successful grasps under uncertain conditions.

By comparing multiple candidate grasps, GQCNN can choose the grasp with the high-

est predicted success probability, leading to robust performance in various practical scenarios. Despite its computational complexity, GQCNN's ability to predict grasp success probability has made it an essential tool in the field of robot grasping.

2.4 Large Language Models for Task Planning

Large Language Models (LLMs) have revolutionized the field of natural language processing (NLP). These models, such as Generative Pre-training Transformer (GPT) [14] and Bidirectional Encoder Representations from Transformers (BERT) [25], are trained on vast amounts of text data, enabling them to generate human-like text and understand context in a way that was previously unachievable. They can perform a variety of tasks, including translation, question answering, and even writing essays. The key to their success lies in their ability to learn patterns in data and apply this knowledge to new, unseen data. However, despite their impressive capabilities, they also have limitations. For instance, they can sometimes generate incorrect or nonsensical responses, and they require significant computational resources to train and operate.

Traditionally, optimizing task plans for robots involves either minimizing the number of actions or the total plan cost, depending on whether action costs are considered. The emergence of Large Language Models (LLMs), such as Google's Bard, OpenAI's ChatGPT [14], and Meta's LLaMA [26], have reshaped the landscape of AI, including task planning for robots [27]. We categorize the LLM-based planning methods into the two groups of *LLM-Native Planning Methods* and *LLM-Aided Planning Methods*, where the former does not rely on external knowledge and the latter does. While LLMs are not strong in numerical reasoning (and hence optimization), the incorporation of LLMs improves the

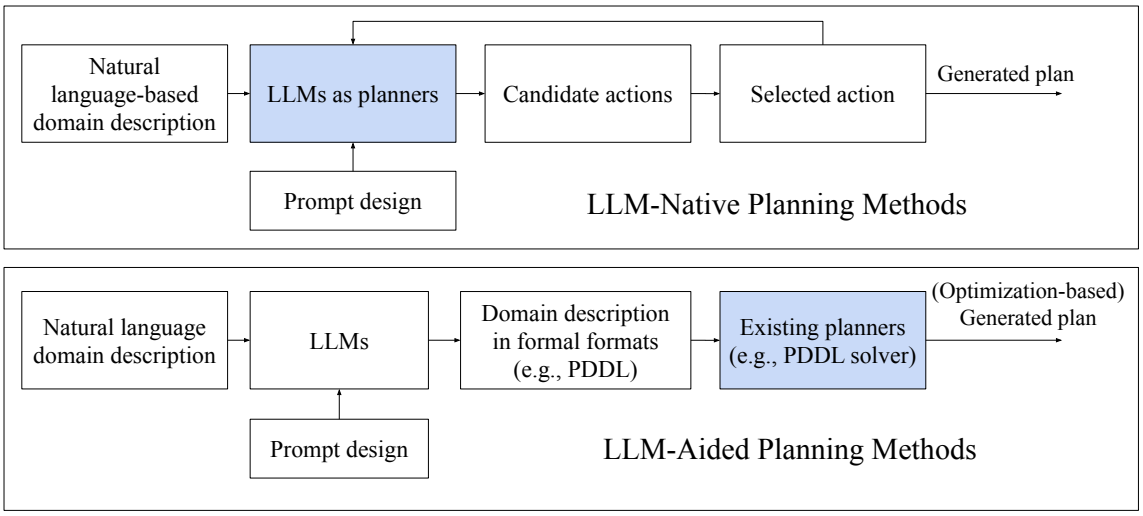


Figure 2.1: There are two methodologies for task planning using LLMs. The key difference lies in the role of LLMs: the first method uses LLMs for planning, while the second method employs LLMs to generate domain descriptions for other existing planning systems.

capabilities of natural language understanding, the acquisition of world knowledge, and commonsense reasoning. Such capabilities make LLM-based task planners more friendly to end users, and better deal with tasks in open-world scenarios.

LLM-Native Planning Methods: One method to leverage LLMs for task planning involves directly generating plans from LLMs by providing a domain description. This can be done either in a one-shot way or iteratively. These methods primarily focus on prompt design for effectively communicating with LLMs, and the grounding to specific domains and robot skills. Several systems have made efforts in this field. Huang et al. [28] to generate candidate actions and propose tools to improve their executability, such as enumerating all permissible actions and mapping the model's output to the most semantically similar action. However, this method has limitations in creating consistent task plans, and some plans cannot be executed by robots. Building upon this, SayCan [29] enables robotic plan-

ning using affordance functions that determine action feasibility and respond to natural language requests such as "deliver a Coke". An advanced approach, named Inner Monologue, developed by Huang et al. [30], integrates environmental feedback for task planning and situation handling. Previously, methods typically generated task plans in text form. Singh et al. [31] develop a system, called ProgPrompt, which employs programmatic LLM prompts to generate task plans and manage situations, by verifying the preconditions of the plan and reacting to failed assertions with suitable recovery actions. Employing code as the framework for high-level planning provides a few benefits. It allows for the expression of complex functions and feedback loops. These loops effectively process sensory outputs and enable the parameterization of control primitives within APIs [32]. Apart from planning for robots, there is also research on whether LLMs can act as universal planners. They could potentially create programs that efficiently generate plans for various tasks within the same domain [33].

LLM-Aided Planning Methods: Prior to the development of Large Language Models (LLMs), various tools existed for robot task planning, but they had scalability limitations. For example, defining domain knowledge in PDDL demanded significant time from human experts. The advent of LLMs offers a way to augment these traditional planners by supplementing knowledge, thereby improving their performance and enabling more natural language interaction. There are a few classical ways of integrating LLMs and task planners. First, a series of studies were conducted to explore the conversion of natural language descriptions of planning tasks into formal formats like PDDL or LTL. LLMs complete these transformations, playing a crucial role in the process. These translated specifications are then used in existing planning systems. For example, Liu et al. [34] and Xie et al. [8]

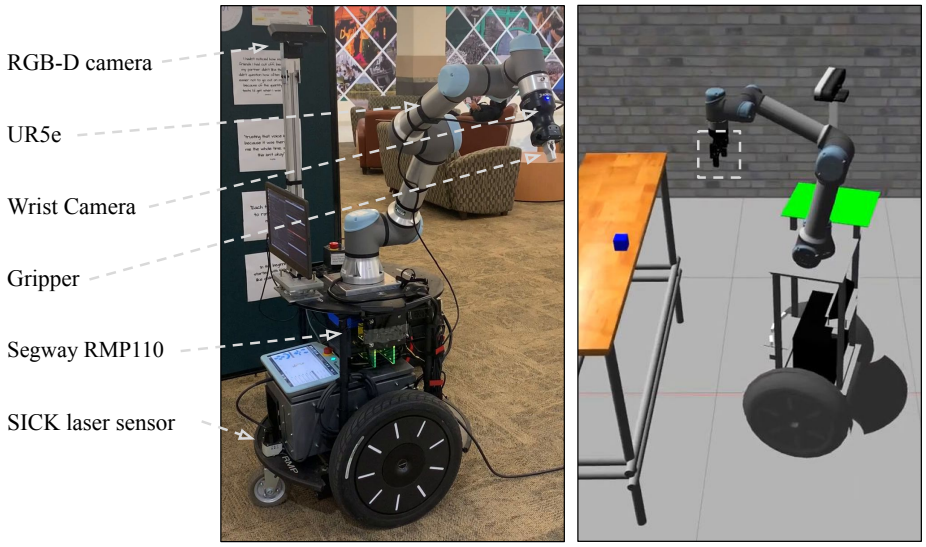


Figure 2.2: The mobile manipulator in both its actual and simulated forms

create optimality-based task-level plans with the PDDL planner, translating natural language inputs into PDDL problems. Second, one can dynamically extract commonsense knowledge from LLMs, enhancing PDDL's action knowledge for planning and situational handling [10]. Third, Zhao et al. [35] utilize LLMs to build world models, and perform heuristic policy in search algorithms such as Monte Carlo Tree Search (MCTS), which uses the common-sense knowledge provided by LLMs to generate possible world states, thus facilitating efficient decision-making.

2.5 Mobile Manipulation Platform

My research mainly utilized a mobile manipulator. Figure 2.2 presents it in both its actual and simulated forms. 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, as detailed in Chapter 7.

The robot's software operates on the Robot Operating System (ROS) [36] and is built upon the publicly available Building Wide Intelligence codebase [37]. This allows the robot to navigate and manipulate objects, including grasping and ungrasping, using MoveIt [38] and GGCNN.

Additionally, we have developed a simulation platform based on the Gazebo physics engine [39]. A difference is the use of the robotiq 2F-85 gripper (marked in dashed box in Figure 2.2) in the simulation, as the urdf model for the Hand-E is unavailable.

3 Planning under Uncertainty with Full Observability

3.1 Introduction

Autonomous driving technologies have the great potential of reshaping urban mobility in people's daily life [40, 41, 42]. To be deemed useful, autonomous vehicles[1] must be time-efficient in accomplishing service tasks, which frequently requires symbolic actions such as "*Merge left, go straight, turn left, and park right"*, while at the same time ensuring safety in executing such actions on the road [43, 44, 45].

Generally, autonomous vehicles need to plan at the *task level* to compute a sequence of symbolic actions toward fulfilling service requests from people. In this process, how the actions are implemented in the real world is out of consideration at the task level. At the same time, vehicles must plan at the *motion level* to compute continuous trajectories, and desired control signals (e.g., for steering, accelerating, and braking) to implement the symbolic actions. While the task planner hopes that all the symbolic actions can be implemented by the vehicles, there is the safety concern that must be considered at the motion level. For instance, lane-changing behaviors can be dangerous in heavy traffic. Figure 3.1 shows two situations that are dangerous (**Left**) and safe (**Right**) to a vehicle, respectively.

Although task planning (frequently referred to as behavior planning in autonomous

[1] Autonomous vehicles are referred to as vehicles for simplicity in the following sections of this work.

Figure 3.1: **Left:** a risky situation for the vehicle (blue) to merge left due to the busy traffic. **Right:** a safe situation for the vehicle to merge left. The goal of TMPUD (ours) is to enable the motion level to take symbolic actions from, and communicate safety to the task level toward efficient and safe autonomous driving behaviors.

driving [46]) and motion planning have been individually conducted in autonomous driving, there is little research from the literature focusing on the interaction between task and motion levels. *There is the critical need of developing algorithms to bridge the gap between task planning and motion planning to help vehicles improve the task-completion efficiency while ensuring the safety of driving behaviors.*

The robotics community has studied the integration of task and motion planning, mostly in manipulation domains [47, 48, 49, 50]. In comparison to those domains, autonomous driving algorithms must consider the uncertainty from the ego vehicle, and the surrounding objects (including other vehicles) on the road. The uncertainty must be quantitatively evaluated at the motion level, and taken into consideration for planning at the task level. For instance, *when the left lane is busy and missing the next crossing does not introduce much extra distance, the task planner should avoid forcing the vehicle to merge left.* Such behaviors are possible, only if the interactions between task and motion levels are enabled.

In this work, we develop **Task-Motion Planning for Urban Driving (TMPUD)** for efficient and safe autonomous urban driving. TMPUD, for the first time, enables the inter-

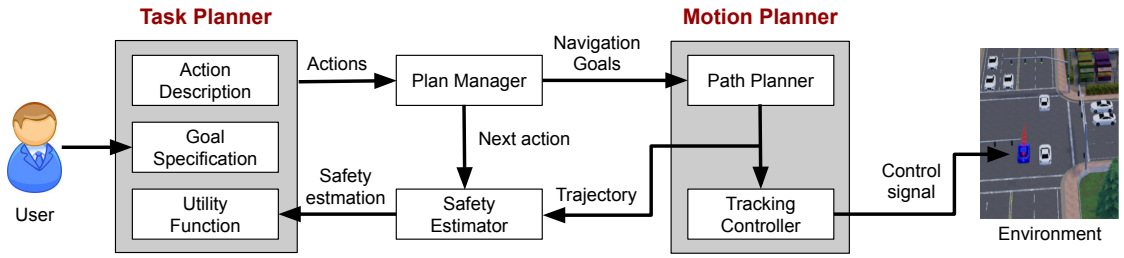


Figure 3.2: An overview of algorithm TMPUD that consists of four components, i.e., *task planner*, *plan manager*, *motion planner* and *safety estimator*. *Task planner* includes components of goal specification, action description, and utility function, where users' service requests are received by the *goal specification* component. Task planner computes a sequence of symbolic actions that are passed to the *plan manager*. The plan manager generate navigation goals (i.e., a pair of two poses) to *path planner*, which is then used for computing a continuous trajectory for connecting 2D poses. The trajectory will be used in the two components of *safety estimator* and *tracking controller*. *Safety estimator* uses this trajectory to estimate actions' safety levels, and then the utility function in task planner can be updated accordingly. *Tracking controller* computes the desired control signals to drive the vehicle to follow the trajectory from the path planner.

action between task and motion planners through enabling the motion-level safety estimation and task-level replanning capabilities. The **contribution of this research** is two-fold, including the new safety estimator, and the TMPUD algorithm. We have implemented and evaluated TMPUD using CARLA, an autonomous driving platform for simulating urban driving scenarios [11]. Results suggest TMPUD improves both safety and efficiency, in comparison to two baseline methods from the literature [51, 47].

3.2 Related Work

We summarize three research areas that are the most relevant to this research, namely motion planning in autonomous driving, task planning in autonomous driving, and the integration of task and motion planning.

Motion-Level Planning for Autonomous Driving: Safety is of the most importance at the

motion level, and highly relies on the motion-level controllers. Early research in robotics (mostly on manipulation problems) has developed a "safe set" algorithm to avoid unsafe situations in human-robot interactions [52], where it offers a theoretical guarantee of safety. That algorithm has been improved to further account for the uncertainty from the real world [53], where both efficiency and safety were modeled in human-robot interface scenarios. Those methods focused on robot manipulation domains, where it is frequently assumed the acting agent being the only one that makes changes in the world, and hence are not applicable to autonomous driving domains.

More recently, within the autonomous driving context, researchers have developed a series of learning and decision making methods to enable motion planners to learn safe behaviors [54, 55, 56, 57, 58, 59]. The above-mentioned methods (in robotics and autonomous driving) mainly focused on motion-level behaviors, and did not look into how motion-level behaviors can be sequenced at the task level to accomplish complex driving tasks. By contrast, TMPUD supports the interaction between task and motion levels, and aims at improving both safety and efficiency of autonomous driving behaviors.

Task-Level Planning for Autonomous Driving: Task planning has been applied to autonomous driving. For instance, one of the earliest works on this topic demonstrated that task planning techniques enable vehicles to complete complex tasks, such as to avoid temporary roadblocks [51] (we use this approach as a baseline in experiments). However, their work did not consider costs of driving behaviors, and hence performs poorly in task-completion efficiency. Similarly, task planners in [60, 61, 62] have *no interaction* with the motion planner, and safety was not modeled in generating the driving behaviors.

Moving forward, more recent research has enabled vehicles to periodically verify the task sequences and motion trajectories against the actual traffic situation [12]. In case of possible dangers detected at the motion level, re-planning is triggered at the task level. The main limitation of their work is that the triggering is deterministic, and highly depends on a safety threshold. The threshold must be set beforehand to ensure safety, which frequently produces over-conservative behaviors, and significantly reduces the task-level efficiency.

Task and Motion Planning: Researchers have integrated task and motion planning in robotics, where the primary domain is robot manipulation [63, 64, 65, 66, 47]. Research on manipulation is mostly concerned with the motion-level feasibility, e.g., in grasping and ungrasping behaviors, and accomplishing high-level tasks, such as stacking objects. Those methods did not consider the uncertainty from other agents (e.g., vehicles on the road). As a result, their systems produce over-optimistic (and hence risky) behaviors, assuming no other agents making changes in the world, and are not applicable to autonomous driving domains.

A journal work has surveyed frameworks for autonomous driving [46], including works that plan at both task and motion levels. However, their motion planners do not provide any feedback to the task level, except for infeasible actions. In comparison, our TMPUD algorithm supports motion-level safety evaluation, and enables the task planner to dynamically adjust high-level plans to account for current road conditions toward accomplishing long-term driving tasks.

3.3 Background

We very briefly summarize task planning and motion planning, the two building blocks of this research.

Task Planning: A task planning domain is specified by D^t, including a set of states, S, and a set of actions, A. We assume a factored state space such that each state $s \in S$ is defined by the values of a fixed set of variables; each action $a \in A$ is defined by its preconditions and effects. A utility function maps the state transition to a real number, which takes both *cost function* $Cost(\langle s,a,s' \rangle)$ and *safety function* $Safe(\langle s,a,s' \rangle)$ into account. Specifically, the cost and safety functions respectively reflect the cost and safety of conducting action a in state s.

Given domain D^t and a task planning problem, we want to compute a plan $p \in P$, starting from an initial state $s^{init} \in S$ and finishing in a goal state $s^g \in S$. A plan p consists of a sequence of transitions that can be represented as: $p = \langle s_0, a_0, \cdots, s_{N-1}, a_{N-1}, s_N \rangle$, where $s_0 = s^{init}$, $s_N = s^g$ and P denotes a set of satisfactory plans. Task planner P^t can produce an optimal plan p^* among all satisfactory plans, where γ is a constant coefficient and $\gamma > 0$.

$$p^* = \arg\min_{p \in P} \sum_{\langle s,a,s' \rangle \in p} [Cost(\langle s,a,s' \rangle) + \frac{\gamma}{1 + e^{Safe(\langle s,a,s' \rangle)-1}}]$$

Motion Planning: A motion planning domain is specified by D^m, where we directly search in 2D space constrained by the urban road network. Some parts of the space are designated as free space, and the rest are designated as obstacles. The 2D space is represented as a region in Cartesian space such that the position and orientation of the vehicle can be

uniquely represented as a pose, denoted by x.

Given domain D^m, a motion planning problem can be specified by an initial pose x^i and a goal pose x^g. The motion planning problem is solved by a motion planner P^m consisting of path planner and tracking planner into two phases. In the first one, a path planner computes a collision-free trajectory ξ connecting pose x^i and pose x^g taking into account any motion constraints on the part of the vehicle with minimal trajectory length. In the second one, a tracking controller computes desired control signals to drive the vehicle to follow the computed trajectory. Due to the fundamental difference between representations at task and motion levels, in line with past research [63, 66, 47, 50], we use a *state mapping function*, $f : X = f(s)$, to map the symbolic state s into a set of feasible poses X in continuous space, for motion planner to sample from. We assume the availability of at least one pose $x \in X$ in each state s, such that the vehicle is in the free space of D^m. If it is not the case, the state s is declared infeasible.

3.4 Algorithms

In this section, we present our main contribution of this research, including two algorithms for safety estimation, and efficient and safe urban driving.

3.4.1 Safety Estimation

Safety estimation aims at computing the safety level, $Safe(\langle s, a, s' \rangle)$, of the motion-level implementation of a symbolic action $\langle s, a, s' \rangle$. The goal of computing the safety value is to enable the task planner to incorporate the road condition into the process of sequencing high-level actions toward accomplishing complex driving tasks.

Algorithm 1 Safety Estimation

Require: Symbolic action $\langle s,a,s'\rangle$, state mapping function f, motion planner P^m, control operation sets Δ and Θ

1: Sample initial and goal poses, $x \leftarrow f(s)$ and $x' \leftarrow f(s')$, given action $\langle s,a,s'\rangle$, and f.
2: Compute a collision-free trajectory, ξ^E, using P^m, where $\xi^E(t_1)=x, \xi^E(t_2)=x'$, and $[t_1,t_2]$ is the horizon
3: Predict trajectory ξ_i^S for the ith surrounding vehicle, where $i \in [1,\cdots,N]$, and $[t_1,t_2]$ is the horizon
4: **while** for each vehicle V_i **do**
5: Compute safe control set $U_i^S(t)$ between the ego vehicle and vehicle V_i at time $t \in [t_1,t_2]$, where $U_i^S(t) \subset \Delta \times \Theta$ and $t = t_1 + \omega \times i, i \leq \lfloor \frac{(t_2-t_1)}{\omega} \rfloor$
6: Sample M elements $\langle \delta, \theta \rangle$ randomly from set $\Delta \times \Theta$ and compute the probability $o_i(t)$ of the elements falling in set $U_i^S(t)$
7: Convert a list of estimated safety values, $\{o_i(t)\}$, into a scalar value o_i^* using Eqn. 3.1
8: **end while**
9: **return** $min\{o_i^*, i = 1,\cdots,N\}$

Terminology: To perform symbolic action $\langle s,a,s'\rangle$, we use a motion planner to compute a sequence of continuous control signals, i.e., acceleration $\delta \in \Delta$ and steer angle $\theta \in \Theta$, to drive the vehicle following the planned trajectory, while ensuring no collision on the road. Sets Δ and Θ denote the operation specification of the controller, which generally depends on the adopted motion planner and the ego vehicle itself. Let $U_S(t)$ (mathematically $U_S(t) \subset \Delta \times \Theta$) specify a *safe control set* at time t, in which all elements, denoted by $u(t) = \langle \delta, \theta \rangle$, are *safe* for an ego vehicle to perform at time t. Intuitively, the size of safe control set U_S reflects the safety level. For instance, when $|U_S|$ is very small, meaning that very few control signals are safe, the vehicle can only be operated in very particular ways, indicating the safety level in general is low. Accordingly, *we use the probability of elements sampled from set* $\Delta \times \Theta$ *being located in the safe set* U_S *to represent the safety value of action* $\langle s,a,s'\rangle$.

Safety Estimation Algorithm: Algorithm 1 summarizes the procedure of our safety es-

timation algorithm. The input includes symbolic action $\langle s, a, s' \rangle$, stating mapping function f, motion planner P^m consisting of path planner and tracking controller, and the controller's operation specification sets Δ and Θ. The output is the estimated safety value $Safe(\langle s, a, s' \rangle) \in [0.0, 1.0]$.

Lines 1-3 aim to obtain the short-period trajectories of the ego and surrounding vehicles, where $V_i, i \in [1, \cdots N]$, is the ith vehicle within the ego vehicle's sensing range. More specifically, we first sample a pair of feasible initial and goal poses for the symbolic actions using the state mapping function (Line 1). Taking these two poses as input, the motion planner then computes a continuous trajectory for our ego vehicle for a short period of time $[t_1, t_2]$ (Line 2), where t_1 is the current time, and $t_2 = t_1 + T$ indicates the time horizon of the ego vehicle. We predicate surrounding vehicles' trajectories, assuming their linear and angular speeds being stationary (Line 3), though there are more advanced methods [67, 68], which is beyond the scope of this research.

Lines 4-8 present a control loop that computes the safety estimation between the ego vehicle and the surrounding vehicles $V_i,$ wehre $i \in [1, \cdots, N]$, given that the ego vehicle is performing action $\langle s, a, s' \rangle$ at the motion level. We compute a safe control set $U_i^S(t)$, similar to [57], that includes all safe control signals with regard to vehicle V_i at time t (Line 5). Parameter ω controls the sampling interval. In Line 6, we randomly sample M elements from the set $\Delta \times \Theta$, and compute probability $o_i(t)$ of the sampled elements falling in set $U_i^S(t)$. We convert a list of values of safety estimation $\{o_i(t)\}$ into a single value o_i^* using eqn.3.1, where *max* and *mean* are two functions to calculate the maximum and mean value of a list, respectively (Line 7). Although all surrounding vehicles can potentially introduce risks to the ego vehicle, we assume the ego vehicle only considers the most dangerous

vehicle. Accordingly, Line 9 is used for selecting the minimum value, $o_i^*, i \in [1, \cdots, N]$, as the overall safety value:

$$o_i^* = \frac{max_{t \in \mathscr{T}}\{o_i(t)\} + mean_{t \in \mathscr{T}}\{o_i(t)\}}{2} \tag{3.1}$$

where $\mathscr{T} = t_1 + \omega \times i, \;\; 0 \leq i \leq \frac{(t_2 - t_1)}{\omega}$

3.4.2 TMPUD

Our motion planner P^m computes both costs (trajectory lengths) and safety values of the ego vehicle's navigation actions, which have been discussed in Section 3.4.1. Here, we focus on the main contribution of this work on enabling interactive task-motion planning for urban driving.

Terminology: We use s^{init} to represent the initial state of the ego vehicle, and the goal (*service request* from people) is specified using s^g. Our task planner P^t computes a sequence of symbolic actions, and it requires two functions that are initialized and updated within the algorithm, including cost function $Cost$, and safety estimation function $Safe$. Motion planner P^m is used for computing motion trajectories, and generating control signals to move the ego vehicle. The state mapping function f is used for mapping symbolic states to 2D coordinates in continuous spaces.

The TMPUD Algorithm: Algorithm 2 summarizes the procedure of TMPUD. It starts by initializing the cost and safety estimation functions (Lines 1 and 2). Cost function *Cost* is initialized using A star algorithm provided by CARLA, as shown in Line 1. In

Algorithm 2 TMPUD algorithm

Require: Initial state s^i, goal specification s^g, task planner P^t, state mapping function f, motion planner P^m, and safety estimator (Algorithm 1)

1: Initialize cost function $Cost$ with sampled poses $x \in f(s)$: $Cost(\langle s,a,s' \rangle) \leftarrow A^*(x,x')$
2: Initialize safety estimation $Safe(s,a,s') \leftarrow 1.0$
3: Compute an optimal task plan p using $Cost$ and $Safe$ functions: $p \leftarrow P^t(s^{init}, s^g, Cost, Safe)$, where $p = \langle s^{init}, a_0, s_1, a_1, \cdots, s^g \rangle$
4: **while** Plan p is not empty **do**
5: Extract the first action of p, $\langle s,a,s' \rangle$, and compute safety value μ using **Algorithm 1**
6: Update $Safe$ function: $Safe(\langle s,a,s' \rangle) \leftarrow \mu$ and $Cost$ function: $Cost(\langle s,a,s' \rangle) \leftarrow A^*(x,x')$
7: Generate a new plan: $p' \leftarrow P^t(s, s^g, Cost, Safe)$
8: **if** $p' == p$ **then**
9: $x' \leftarrow f(s')$
10: **while** $x \,!= x'$ **do**
11: Call motion planner $\langle \delta, \theta \rangle \leftarrow P^m(x,x')$
12: Execute the control signal $\langle \delta, \theta \rangle$
13: Update the vehicle's current pose x
14: **end while**
15: Remove the tuple $\langle s,a \rangle$ from plan p
16: **else**
17: Update current plan $p \leftarrow p'$
18: **end if**
19: **end while**

Line 2, TMPUD optimistically initializes the safety estimation function by setting 1.0 to all actions, indicating all task-level actions are completely safe. After that, an optimal task plan, $p^* = \langle s^{init}, a_0, s_1, \cdots, s^g \rangle$, is computed in Line 3. The head and tail elements of the plan, s^{init} and s^g, correspond to the initial and goal poses respectively.

Lines 4-19 form TMPUD's main control loop that enables the interaction between task and motion planners. The loop's termination condition is the task-level plan being empty, i.e., the goal has been achieved (Line 4). Specifically, TMPUD estimates the safety level, μ, of action $\langle s,a,s' \rangle$ (Line 5). Functions $Safe$ and $Cost$ are updated using μ and $A*$ search in Line 6. Then a new optimal plan p' is computed in Line 7. Lines 8-18 is for plan

monitoring and action execution. If the task planner suggests the same plan (Line 8), the vehicle will continue to execute action a at the motion level. The goal state is sampled from state mapping function in Line 9. Line 10-14 is a loop to execute the action. Specifically, the motion planner will compute and execute a desired control signal $\langle \delta, \theta \rangle$ repeatedly until the vehicle reaches the goal pose (Line 10). The vehicle's current pose x will be updated after each execution (Line 13). After completing the operation, the tuple $\langle s, a \rangle$ will be removed from the plan p (Line 15). On the contrary, if the task planner suggests a new plan p' different from the plan p, the *currently optimal* p' will replace the *non-optimal* plan p (Line 17).

3.4.3 Algorithm Instantiation

Task Planner: Our task planner P^t is implemented using Answer Set Programming (ASP), which is a popular declarative language for knowledge representation and reasoning, and ASP has been used for task planning [20, 69, 50, 70]. For example, predicate `leftof(La1,La2)` can be used to specify lane `La1` being on the left of lane `La2`. We model five driving actions, including `mergeleft`, `mergeright`, `forward`, `turnleft`, and `turnright`. For instance, action `mergeright` can be used to help the vehicle merge to the right lane, where constraints, such as "`changeright`" is allowed only if there exists a lane on the right, have been modeled as well.

Motion Planner: At the motion level, path planner firstly generates a desired continuous trajectory with the minimal traveling distance using $A*$ search. The trajectory includes a set of waypoints (each in the form of a pair of $x-y$ coordinate and orientation), and the

trajectory is delivered to the tracking controller, along with the vehicle's current pose and speed. The controller uses a proportional-integral-derivative (PID) controller [71] to generate control signals, e.g., for steering, throttle, and brake. PID controller is very popular due to its simplicity, flexibility, and robustness.

3.5 Experiments

We use CARLA, an open-source 3D urban driving simulator [11] in this research. CARLA (Car Learning to Act) has been developed to support development, training, and validation of autonomous driving systems. Compared to other simulation platforms, e.g., [72, 39], CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely.

Illustrative Example: Figure 3.3 presents an illustrative example. TMPUD starts with using our optimal task planner to compute *Plan A*. The vehicle takes the first symbolic action from *Plan A* (trajectory in blue color), and executes the action using our motion planner. Getting close to *Area 1*, the vehicle plans to *merge left*. However, the safety estimator at the motion level reports a low safety value in *Area 1*. This computed safety value is incorporated into task planner, where the task planner integrates the safety value into its cost function, and re-computes an optimal plan, *Plan B*. Different from *Plan A*, *Plan B* suggests the vehicle to *go straight*, and *merge left* in *Area 2*. In this trial, the vehicle was able to follow *Plan B* all the way to the goal. TMPUD enabled the vehicle to avoid the risky behavior of merging left in *Area 1* without introducing extra motion cost. A demo video is provided on YouTube.[2]

[2] `https://youtu.be/8NHQYUqMyoI`

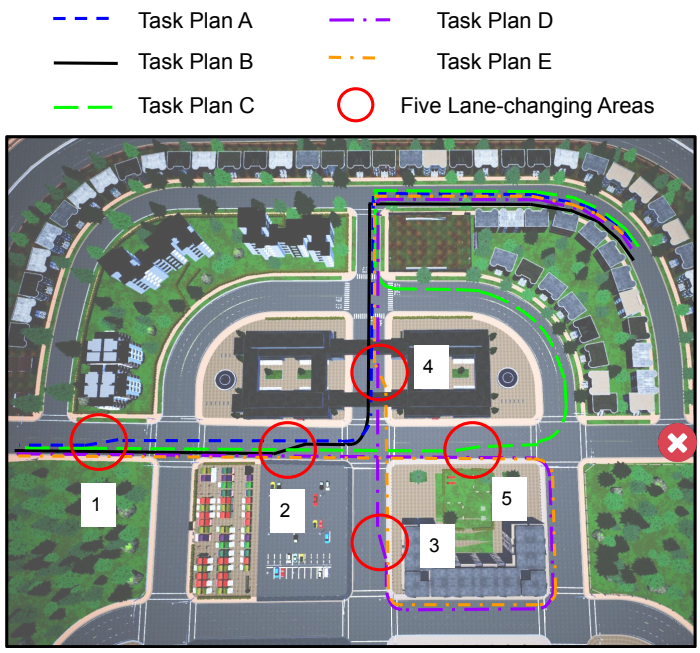


Figure 3.3: An illustrative example, where the vehicle is tasked with driving from the very left to the top-right area. The vehicle needs to compute plans at both task and motion levels. The vehicle starts with executing *Plan A* (blue color). While it is getting close to *Area 1* (a red circle), the motion-level safety estimator reports a low safety value based on the local road condition. This computed safety value is incorporated into task planner's cost function. Using the updated cost function, the task planner re-computes an optimal plan (*Plan B*), and suggests the vehicle to *go straight* and *merge left in* Area 2. The interaction between task and motion levels, supported by TMPUD, enables the vehicle to dynamically adjust its high-level task plans to avoid unsafe behaviors.

3.5.1 Full and Abstract Simulation Platforms

Experiments conducted in CARLA are referred as being in **full simulation**. All vehicles move at a constant speed ($20km/h$) on average. In full simulation, ego vehicle performs the whole plan at the task level in a the presence of other vehicles. We spawn different numbers of vehicles (200 and 120), and refer to traffic of the two environments as being **heavy** and **normal** respectively.

Running full simulation using CARLA is time-consuming, preventing us from conducting large-scale experiments. For instance, results reported in this work are based on

tens of thousands of experimental trials, and *full simulation in this scale would have required months of computation time*. To conduct large numbers of experimental trials, we developed an **abstract simulation** platform, where action outcomes are sampled from precomputed probabilistic world models. Parameters of the world models (for abstract simulation) are learned by repeatedly spawning the ego and surrounding vehicles in a small area, and statistically analyzing the results of the vehicles' interaction.

In particular, we spent the most effort in analyzing the outcomes of "merging lane" actions due to its significant potential risks. We empirically computed the probabilities of the three different outcomes of "merging lane" actions, including "merge", "collide", and "stop". We introduced **two domain factors** into the abstract simulation platform, including **density** and **acceleration**. In high-density environments, the ego vehicle is surrounded by three vehicles, while this number is reduced to one in low-density environments. In high-acceleration environments, surrounding vehicles' acceleration (in m/s^2) is randomly sampled in $[-1.0, 1.0]$, while this range is $[-0.5, 0.5]$ in low-acceleration environments.

3.5.2 Evaluation Metrics and Two Baseline Methods

The goal of TMPUD is to improve task-completion efficiency (to reduce traveling distance), while guaranteeing safety. So, the two most important evaluation metrics are **traveling distance** and **the number of unsafe behaviors**, where unsafe behaviors cause either collisions or force at least one surrounding vehicle to stop (to avoid collisions).

The two baseline methods used in this research are selected from the literature, and referred to as *No-communication* (**No-com**), and *Threshold-based* (**Th-based**). The No-com baseline [51] forces the vehicle to execute all task-level actions at the motion level,

Table 3.1: **Full simulation**: Traveling distance and number of collisions and stops for three algorithms under different traffic conditions (normal and heavy traffic).

	Algorithm		Traveling Distance (m)	Num. of collisions and stops
Normal Traffic	TMPUD		514	0
	Th-based	$\beta = 0.5$	537	0
		$\beta = 0.3$	513	5
		$\beta = 0.1$	478	24
	No-com		426	48
Heavy Traffic	Algorithm		Traveling Distance (m)	Num. of collisions and stops
	TMPUD		530	2
	Th-based	$\beta = 0.5$	545	2
		$\beta = 0.3$	528	7
		$\beta = 0.1$	497	35
	No-com		426	54

while driving behaviors' safety values are not considered. The Th-based baseline [47] enables the motion planner to "reject" a task-level action when its safety value is lower than a threshold β, where a higher (lower) β threshold makes a vehicle more conservative (aggressive). In case of an action being rejected, the task planner will compute a new plan to avoid the risky action. We develop **three versions** of of the Th-based baseline with different β values (0.1, 0.3, and 0.5).

3.5.3 Experimental Results

Results from Full Simulation: Table 3.1 presents the results in comparing TMPUD to the two baseline methods. From the table, we see that, in both road conditions, TMPUD achieved the lowest traveling distance, in comparison to those methods that produced compared safety levels (in terms of the number of collisions and stops). For instance, under normal traffic, only the Th-based baseline with $\beta = 0.5$ was able to completely avoid collisions and stops, but it produced an average traveling distance of $537m$. In comparison,

TMPUD required only $514m$, while completely avoided collisions and stops. Under heavy traffic, TMPUD (again) produced the best performance in safety (based on the number of collisions and stops), while requiring less traveling distance in comparison to the only baseline (*Th-based* with $\beta = 0.5$) that produced comparable performance in safety. The experimental trials (200 for each approach) from full simulation took eight full workdays. We aim at evaluating the performance of TMPUD under different domain factors, requiring a much larger number of trials, which motivated us to conduct experiments using the abstract simulator.

Results from Abstract Simulation:

Figure 3.4 presents the performances of TMPUD and the baseline methods in both traveling distance and the number of unsafe behaviors. The x-axis corresponds to the average traveling distance, and y-axis corresponds to the total number of collisions and stops (both are considered failure cases of driving behaviors). From the four subfigures, we see that TMPUD is the most efficient (x-axis) among those methods that produced comparable performances in safety (y-axis), except that *Th-based* ($\beta = 0.5$) produced slightly less unsafe behaviors (but it performed poorly in efficiency).

There are a few side observation. Not surprisingly, *No-com* produced the worst performance of in safety (y-axis), though its traveling distance remains the lowest. This is because, using *No-com*, the vehicle blindly executes task-level actions while unrealistically believing driving behaviors are always safe. The *Th-based* baseline's performance depends on its safety threshold (β), where a greater value produces safer but less efficient behaviors. The results support our claim that TMPUD improves vehicles' task-completion efficiency, while ensuring safety in different road conditions.

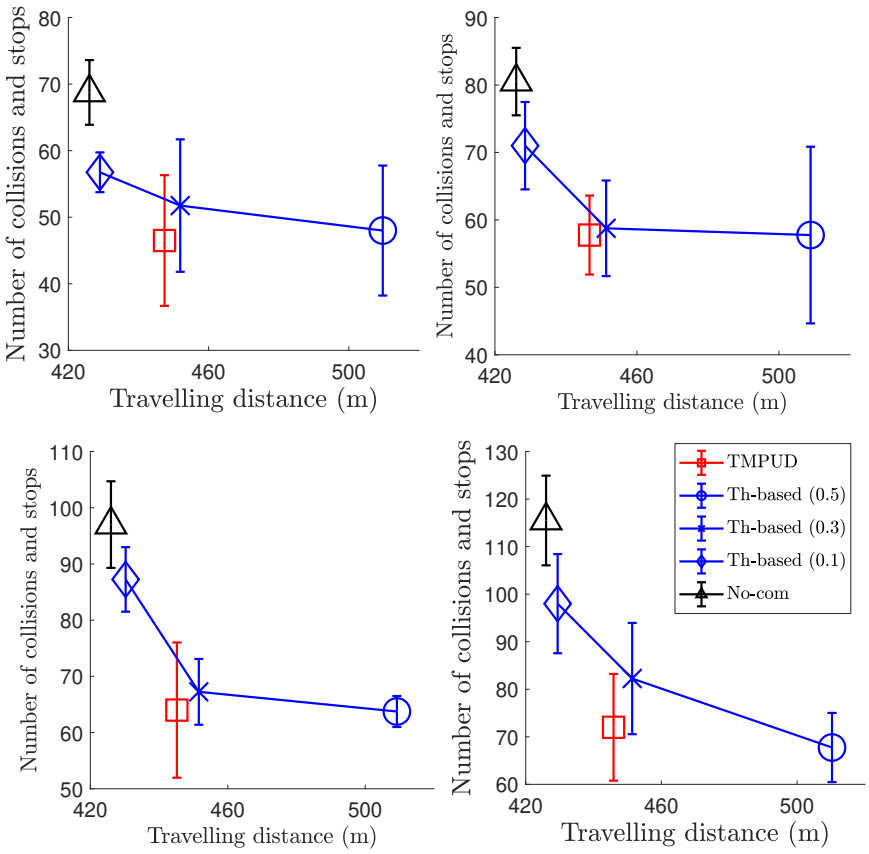


Figure 3.4: **Abstraction simulation:** the overall performances of TMPUD and two baseline methods. The x-axis represents the average traveling distance, and the y-axis represents the total number of collisions and stops. The four subfigures correspond to four different road conditions. The road conditions, from left to right, are **low-density and low-acceleration**, **low-density and high-acceleration**, **high-density and low-acceleration**, **high-density and high-acceleration**. Under each road condition, we evaluate each algorithm using 4000 trials. We did batch-based evaluations with four batches for significance analysis, where each batch includes 1000 trials.

4 Planning under Uncertainty with Partial Observability

4.1 Introduction

Self-driving cars are changing people's everyday lives. Narrowly defined autonomous driving technology is concerned with point-to-point navigation and obstacle avoidance [73], where recent advances in perception and machine learning have made significant achievements. In this work, we are concerned with urban driving scenarios, where vehicles must follow traffic rules and social norms to perform driving behaviors, such as merging lanes and parking on the right. At the same time, the vehicles need to fulfill service requests from end users. Consider the following scenario:

> *Emma asks her autonomous car to drive her home after work. On her way home, Emma needs to pick up her kid Lucas from school, stop at a gas station, and visit a grocery store. In rush hour, driving in some areas can be difficult. Lucas does not like the gas smell, but he likes shopping with Emma.*

The goal of Emma's autonomous car is to efficiently and safely fulfill her requests while respecting the preferences of Emma (and her family). We say a service request is *complex*, if fulfilling it requires the vehicle to visit two or more points of interest (POIs), such as a

gas station and a grocery store, each corresponding to a driving task. Facing such a service request, a straightforward idea is to first sequence the driving tasks of visiting different POIs, and then perform behavioral and motion planning to complete those tasks. However, this idea is less effective in practice, because of the unforeseen execution-time dynamism of traffic conditions. For instance, the vehicle might find it difficult to merge right and park at a gas station because of unanticipated heavy traffic. This observation motivates this work that leverages visual perception to bridge the communication gap between different decision-making layers for urban driving.

In this work, we develop Grounded Layered Autonomous Driving (**GLAD**), a planning framework for urban driving that includes three decision-making layers for service, behavior, and motion respectively. The service (**top**) layer is for sequencing POIs to be visited in order to fulfill users' service requests. User preferences, such as "*Lucas likes shopping with Emma*", can be incorporated into this layer. The behavior (**middle**) layer plans driving behaviors, such as "merge left" and "drive straight". The motion (**bottom**) layer aims to compute trajectories and follow them to realize the middle-layer behaviors. GLAD is novel in its bidirectional communication mechanism between different layers. For example, the bottom layer reports motion cost estimates up to the top two layers for plan optimization. The safety estimates of different driving behaviors (middle layer) are reported to the top layer, and the safety estimation is conditioned on the motion trajectories in the bottom layer. An overview of GLAD is presented in Figure 4.1.

"Grounding" is a concept that was initially developed in the literature of symbolic reasoning [74]. In this work, the vehicle's behavioral planner (middle layer) relies on symbolic rules, such as "*If the current situation is safe, a merge left behavior will move a vehicle to*

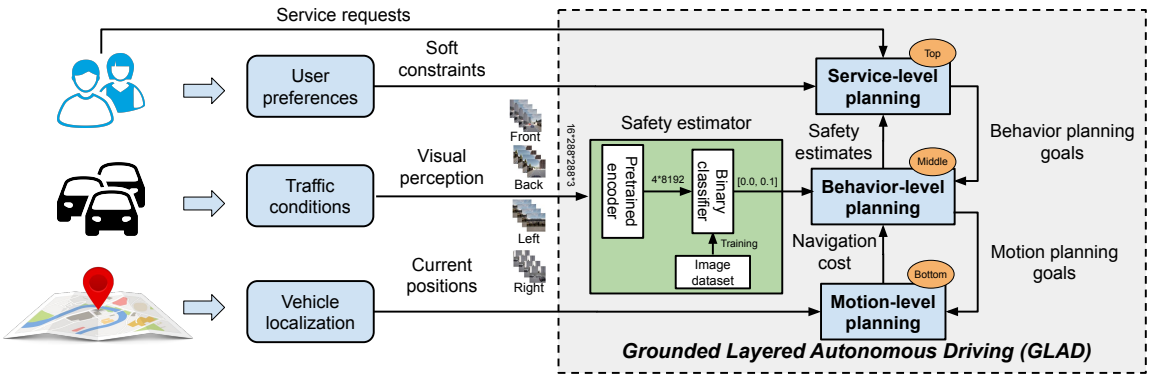


Figure 4.1: Overview of the GLAD planning framework for complex driving tasks in urban scenarios. GLAD consists of three decision-making layers about fulfilling service requests, sequencing driving behaviors, and computing motion trajectories respectively. GLAD is a visually grounded planning framework, because the safety levels of driving behaviors are evaluated using computer vision.

the lane on its left." While classical planning methods assume perfect information about "X is safe," an autonomous vehicle needs its perception algorithms to visually ground such symbols in the real world. We used the CARLA simulator [11] to collect a dataset of 13.8k instances, each including 16 images, for evaluating the safety levels of driving behaviors. Learning from our gathered dataset enables GLAD to visually ground symbolic predicates for planning driving behaviors. We have compared GLAD with baseline methods [12, 13] that support decision-making at behavioral and motion levels. Results show that GLAD produced the highest overall utility compared to the baseline methods.

4.2 Related Work

Service agents sometimes need more than one action to fulfill service requests. *Task planning* methods aim to sequence symbolic actions to complete such complex tasks. There are at least two types of task planning, namely automated planning [75, 76] and planning

under uncertainty [77], that can be distinguished based on their assumptions about the determinism of action outcomes. *Automated planning* systems compute a sequence of actions that lead transitions to a goal state. Assuming non-deterministic action outcomes, *planning under uncertainty* algorithms model state transitions using Markov Decision Processes (MDPs) and compute a policy that maps the current state to action. Other than task planning, autonomous vehicles need *motion planning* algorithms [22, 78] to compute motion trajectories and generate control signals to follow those trajectories. This work involves both task planning and motion planning. Next, we discuss how those different planning paradigms have been used in driving and robotics literature.

Planning for Autonomous Driving: Within the context of planning in autonomous driving, the majority of research has been focusing on computing trajectories connecting a vehicle's current position to its desired goal [79, 80, 56, 81, 82, 83, 84]. For instance, Hu et al. (2018) developed an approach that generates an optimal path in real time as well as the appropriate acceleration and speed for the autonomous vehicle in order to avoid both static and moving obstacles [79]. Those works concentrated on the motion planning and control problems in autonomous driving, whereas we further include task-level planning for driving behaviors and fulfilling service requests.

Driving behaviors, such as merging lanes and turning, have been modeled as a set of symbolic actions [62, 51, 85, 86, 87, 88]. Those works focused on task-level, discrete-space behavioral planning, and did not consider safety, cost, or both at the motion level. Another difference is that those works did not consider user preferences, whereas we do in this work. Recent research has incorporated state estimation into behavioral planning

in autonomous urban driving. For instance, the work of Phiquepal and Toussaint (2019) modeled urban driving as a partially observable Markov decision process to enable active information gathering for planning [89]. Another example is the work of Ding et al. (2020) that quantitatively estimated safety levels for behavioral planning [13]. In line with those methods, we consider partially observable worlds and compute plans under uncertainty. Beyond that, GLAD has a perception component and leverages egocentric vision for safety evaluation, which improves the real-world applicability of our approach compared with those methods.

Task and Motion Planning in Robotics: Researchers in robotics have developed a variety of task and motion planning (TAMP) algorithms to enable robots to plan to fulfill task-level goals while maintaining motion-level feasibility, as summarized in review articles [1, 90]. The majority of TAMP research concentrated on manipulation domains, where ensuring plan feasibility is key, and efficiency in task completion is secondary [91, 49, 92, 93]. Urban driving tasks tend to be time-consuming, and sub-optimal plans might result in a very long execution time. Different from most TAMP methods, our work incorporates task-completion efficiency in addition to plan feasibility.

There are a few TAMP works that incorporated efficiency into plan optimization [94, 95, 96, 10]. For instance, Zhang et al. (2022) leveraged computer vision towards computing efficient and feasible task-motion plans for a mobile manipulator in indoor environments [96]. In line with those methods, we consider both task-completion efficiency and plan feasibility. Different from those TAMP methods for indoor scenarios, GLAD focuses on urban driving, and further considers user preferences and road safety.

4.3 Problem Statement

Planning-time Input: A service request is specified as hard constraints that can be satisfied using multiple driving behaviors, denoted as cst^{hard}. User preferences are formulated using soft constraints, denoted as cst^{soft}. Violating such a soft constraint introduces a penalty. In our domains, service task specifications and user preferences are provided as part of the problem definition. The hard and soft constraints can be encoded in different ways depending on the syntax of planning languages and systems.

Execution-time Sensory Input: The vehicle, provided with a 2D map, can obtain its current position with respect to the map. In addition, four monocular cameras are mounted on the front, back, left, and right sides of the vehicle. Each observation *IM* is in the form of a sequence of four frames, and each frame includes four images captured from the four cameras installed on the four sides of the vehicle. The size of *IM* is $4 \times 4 \times 288 \times 288 \times 3$, where $288 \times 288 \times 3$ is the image size.

Requirements: The driving agent is given a *task planning system*, denoted as $Plnr^t$, that can be used for sequencing driving behaviors to complete a driving task (e.g., visiting a grocery store). $Plnr^t$ includes seven symbolic driving behaviors, including `mergeleft`, `mergeright`, `turnleft`, `turnright`, `gostraight`, `park`, and `stop`. Each action (corresponding to a driving behavior) is defined by a description of its preconditions and effects. In this work, $Plnr^t$ is constructed in Answer Set Programming (ASP) [20].

The driving agent is also given a 2D *motion planning system*, denoted as $Plnr^m$. The motion planning system can be used for computing collision-free, shortest trajectories that

connect the vehicle's current and goal positions. The provided 2D map contains a set of POIs. Each POI is comprised of its name and a 2D position that can be visited by the vehicle.

Utility: Given a task-motion plan, the utility function considers motion costs, safety of driving behaviors, and user preferences, which are denoted as $Cost()$, $Safe()$, and $Pref()$, respectively. Specifically, $Cost()$ is calculated based on the vehicle's traveling time. $Safe()$ evaluates the safety levels of driving behaviors, and its value is the sum of the safety level μ of each driving behavior in a task plan. In this work, we separately compute the safety level of each driving behavior, and do not consider their interactions. A breach of user preferences cst^{soft} incurs a violation cost to $Pref()$. $Cost()$, $Pref()$, and $Safe()$ generate positive real numbers or zeros.

$$Utility(p) = \mathbb{E}\big(\alpha_0 \cdot Cost(p) + \alpha_1 \cdot Pref(p) + \alpha_2 \cdot Safe(p)\big) \tag{4.1}$$

where p denotes the generated task plan which is implemented by the computed motion trajectories. Each trajectory corresponds to one driving behavior, and the trajectories together form one continuous path. α_0 and α_1 are negative constants, and α_2 is a positive constant.

Format of Algorithm Output: The driving agent computes a task-motion plan that includes a sequence of N driving behaviors, and a sequence of N motion trajectories that are head-to-tail connected, where each trajectory corresponds to one driving behavior. *The driving agent's goal is to compute task-motion plans towards maximizing the overall expected utility presented in Eqn. (4.1).* Next, we present our grounded layered planning

algorithm for complex urban driving tasks.

4.4 Algorithm

In this section, we describe Grounded Layered Autonomous Driving (GLAD), a novel planning algorithm to address the problem of "planning to complete complex urban driving tasks", as defined in Section 4.3, as well as its vision-based safety estimation component.

4.4.1 GLAD Algorithm

Algorithm 3 presents our GLAD algorithm – the main contribution of this work. The input of GLAD includes a task planning system $Plnr^t$, a motion planning system $Plnr^m$, a service request cst^{hard}, user preferences cst^{soft}, observation IM, and the current location loc of our vehicle.

GLAD begins with training our vision-based safety estimator. The estimator predicts the safety level $\mu \in [0.0, 1.0]$ of each driving behavior using observation IM (**Line 8**). GLAD initializes the safety level μ of each driving behavior as 1.0 (**Line 9**), which indicates the vehicle optimistically believes all driving behaviors in a plan are absolutely safe. GLAD then calls the function *OptimalPlan* to compute an optimal task-motion plan p^* (**Line 10**).

In the *while-loop*, the vehicle completes driving tasks in plan p^*, while at the same time plan p^* will be continually updated (**Lines 11∼24**). The *outer if-else* iteratively executes the motion trajectory for the vehicle to visit POIs in plan p^* (**Lines 15∼23**). The *inner if* is used to check if an POI in plan p^* is visited (**Lines 17∼19**). At each iteration in the while-loop, the safety level μ of an driving behavior in plan p^* is estimated using the

Algorithm 3 Grounded Layered Autonomous Driving

1: **Function** *OptimalPlan*(cst^{hard}, cst^{soft}, μ, *loc*)
2: Sequence POIs to fulfill cst^{hard}, where each feasible sequence is denoted as *seq*.
3: Generate all feasible sequences of driving behaviors for each *seq* using $Plnr^{t}$, where each plan is denoted as p.
4: Generate a motion plan for plan p using $Plnr^{m}$ with *loc*.
5: Calculate $Cost(p)$, $Pref(p)$, and $Safe(p)$.
6: Compute an optimal plan p^* from all p's based on Eqn. (4.1).
7: **EndFunction**

Require: $Plnr^{t}$ and $Plnr^{m}$
Input: cst^{hard}, cst^{soft}, *IM*, and *loc*

8: Train a vision-based safety estimator (Section 4.4.2).
9: Initialize μ of each behavior as 1.0.
10: Call function *OptimalPlan* to compute an optimal plan p^*
11: **while** cst^{hard} still has driving tasks **do**
12: Extract the first driving behavior in p^*.
13: Estimate μ of the behavior using the trained estimator with *IM* based on Eqn. (4.2).
14: Call function *OptimalPlan* to compute an optimal plan p' with the updated cst^{hard}, μ, and *loc*.
15: **if** $p^* == p'$ **then**
16: Execute the motion trajectories of plan p^*.
17: **if** an POI in p^* is visited **then**
18: Update cst^{hard} by removing completed the driving task.
19: **end if**
20: Update *loc* and remove the first driving behavior in p^*.
21: **else**
22: Update the optimal plan p^* with p'.
23: **end if**
24: **end while**

trained safety estimator with given *IM* based on Eqn. (4.2) (**Line 13**). A new optimal plan p' is then computed with the updated μ and *loc* (**Line 14**). If $Plnr^{t}$ suggests the same plan (**Line 15**), the vehicle will continue to perform the driving behavior at the motion level (**Line 16**). Otherwise, the *currently optimal* p' will replace plan p^* (**Line 22**). After a driving behavior is executed at the motion level, *loc* will be updated (**Line 20**). Next, we discuss how to compute safety level μ using the developed safety estimator (**Line 13**).

4.4.2 Vision-based Safety Estimator

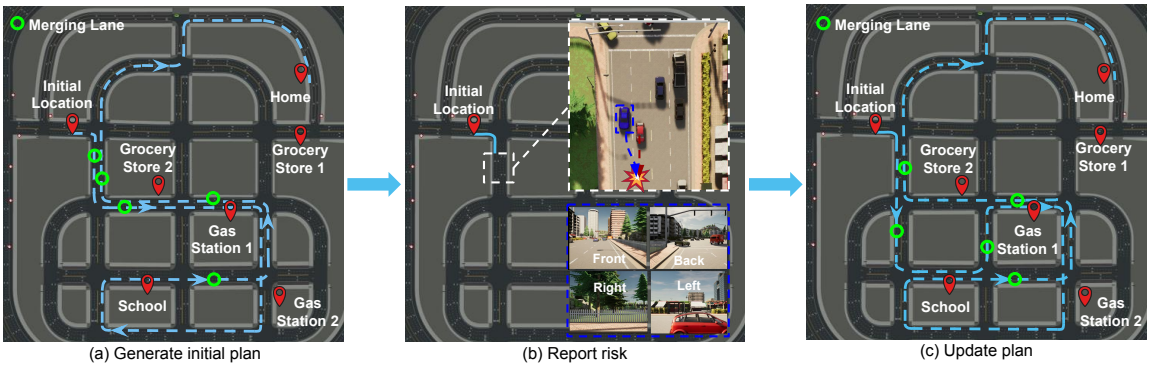
Safety level μ of driving behavior is computed through a learning process using a binary classifier Θ and an encoder Ψ, where the two are separately considered to accommodate different implementations of Θ in our system.

$$\mu = \Theta\big(\Psi(IM)\big) \tag{4.2}$$

where $\Theta()$ learns to generate probability $\mu \in [0.0, 1.0]$ – the safety level of the current driving behavior in this learning and planning framework. $\Psi()$ learns to generate abstract features of road conditions, such as states of surrounding cars and distance to the center of the lane, at the time before executing the driving behavior. To enable this learning process, we used an off-the-shelf pre-trained encoder from the literature [97], which was implemented using ResNet-18 [98]. The learned model $\Psi()$ is capable of outputting a feature vector of the size 32768. Figure 4.2 illustrates GLAD, where the vehicle found it difficult to execute its current plan due to the traffic (visually perceived). Accordingly, GLAD updated the plan to ensure safety and efficiency while respecting user preferences.

4.5 Algorithm Instantiation

Task planning system *Plnr*^{t}: We model seven driving behaviors in ASP [20], such as `mergeleft`. For instance, action `mergeleft(L1,L2)` navigates a vehicle from lane



(a) Generate initial plan

Utility(p) = -893.0 [α_0 • Cost(p) = -1393.0, α_1 • Pref(p) = 0.0, α_2 • Safe(p) = 500.0]

(b) Report risk

Utility(p) = -1843.3 [α_0 • Cost(p) = -1393.0, α_1 • Pref(p) = 0.0, α_2 • Safe(p) = 49.7]

(c) Update plan

Utility(p) = -979.0 [α_0 • Cost(p) = -1479.0, α_1 • Pref(p) = 0.0, α_2 • Safe(p) = 500.0]

Figure 4.2: An illustrative example of GLAD for grounded layered autonomous urban driving. The vehicle's **service task** was to take Emma home after work. On the way home, Emma needed to pick up kid from school, stop at a gas station, and visit a grocery store. To fulfill the service request, our vehicle needed to visit at least four POIs, including School, Grocery Store, Gas Station, and Home. **(a)** GLAD computed a task-motion plan as shown in **blue dashed line**, where at the service level the vehicle planned to visit the following POIs in order: *Gas Station 1*, *School*, *Grocery Store 2*, and *Home*. All POIs are marked with **red pins**. The planned "merge lane" positions are marked with **green circles**. **(b)** Our vehicle (**a blue car**) was preparing to **merge left** in the highlighted area, and observed that there was **a red car** making it unsafe to merge left. **(c)** Based on the computed safety value, GLAD generated a new task-motion plan that helped avoid merging lane in the highlighted area. Although the new plan required a longer traveling distance, it significantly improved driving safety, while considering user preferences. Following the updated plan, the vehicle was able to fulfill the service request.

`L1` to lane `L2`, whose description is shown below:

```
inlane(L2,I+1) :- mergeleft(I), inlane(L1,I),
leftof(L2,L1), step(I), I>=0, I<n.
```

where the right side of `:-` describes the action and its *preconditions*, and the left side is the *effect*. Specifically, its precondition is that the vehicle locates in the lane `L1` at step `I` (i.e., `inlane(L1,I)`), and lane `L2` is on the left of `L1` (i.e., `leftof(L2,L1)`). The corresponding effect is that the vehicle locates in lane `L2` at step `I+1`, i.e., `inlane(L2,I+1)`.

The lanes and their spatial relationships, e.g., `leftof()`, could be easily extracted from a given map. More details about other behaviors are provided in the supplementary materials.

Motion planning system $Plnr^m$: $Plnr^m$ was implemented using the "BasicAgent" function provided by CARLA, which navigates our vehicle to a given target destination from its current position, and also enables our vehicle to follow traffic rules and respect other vehicles. Specifically, $Plnr^m$ is composed of a path planner and a tracking controller. The A^* algorithm is used to compute optimal motion trajectories. A proportional-integral-derivative (PID) controller is applied to generate control signals, e.g., for steering, throttle, and brake.

Safety Estimator was implemented in two ways. One is based on Artificial Neural Networks (ANNs) [99], and the other is based on Support-vector machine (SVM) [100]. The ANN model contains one fully connected hidden layer with 1024 nodes, and the output layer containing 2 nodes is connected to the Softmax function. The negative log-likelihood loss (i.e., NLLLOSS) was applied. We also implemented SVM for binary classification. In this work, we selected Support Vector Classification (SVC), where the parameter "gamma" was set as "auto" and the function "predict_proba" was enabled to get the probability of two classes labels. Given $\Psi(IM)$, the implemented ANN (or SVM) could generate the probability of the current driving behavior being safe (i.e., the safety level $\mu \in [0.0, 1.0]$). We used the following parameters of Eqn. (4.1) for plan optimization: $\alpha_0 = -1.0$, $\alpha_1 = -1.0$, and $\alpha_2 = 500.0$.

4.6 Experiments

We use CARLA [11], an open-source 3D urban driving simulator, for demonstration and evaluation in this work. CARLA has been widely used for autonomous driving re-

search [101]. In particular, we selected Town05 for our experiments, which is featured by its multi-lane roads, and is particularly useful for evaluating complex driving behaviors.

***IM* Dataset Collection:** We built an image dataset to train the safety estimator, where the dataset contains 13.8k instances. Each instance is referred to as *IM*, which includes the *1st*, *2nd*, *4th*, and *10th* frames, as marked in blue box in Figure 4.3. Right after taking an *IM*, the vehicle was forced to merge left (or right) using a predefined motion trajectory. In each trial, the vehicle was given ten seconds to complete the behavior. By the end of the ten seconds, the *IM* was labeled *positive*, if there was a collision with a surrounding vehicle, or there existed another vehicle that was very close to our vehicle (the threshold was 1.0 meter). Otherwise, the instance was labeled *negative*. Figure 4.3 shows two instances (*positive* on the left, and *negative* on the right), where we also present bird views of the two trial. Our dataset is also well balanced, as 46.5% of instances are labeled positive, and the remaining ones are labeled negative. To ensure the diversity of driving scenarios, instances were sampled from 24 different roads. The dataset has been **open-sourced**, where the link for downloading and additional information are provided in the supplementary materials.

4.6.1 Experimental Setup

The NumPy random seed was fixed as 42. Seventy percent of instances in dataset *IM* were used for training, and the remaining instances were used for testing. The training dataset is denoted as D_{train}. There are two training settings, where the non-default is more challenging. In the **default** setting, we used the same roads for training and testing. However, in the **non-default** setting (marked with #), different roads are used for training and testing.

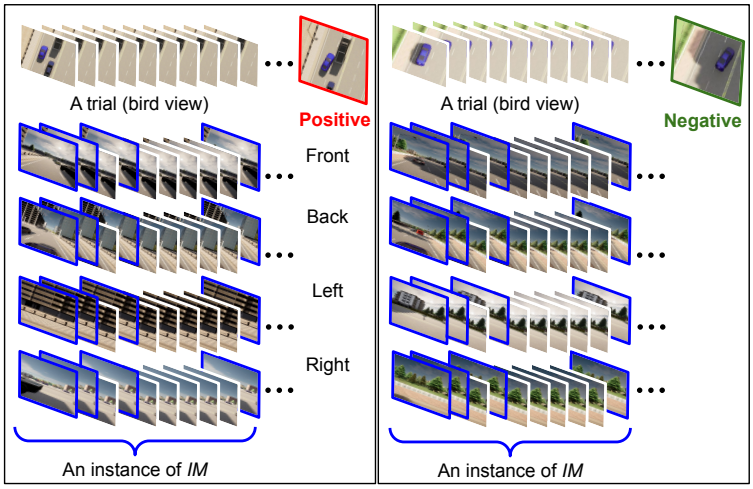


Figure 4.3: Two instances of *IM* with different positive (**Left**) and negative (**Right**) labels in the *IM* dataset, where positive and negative indicate merging lane is unsafe and safe, respectively.

Table 4.1: Utility, cost, penalty caused by violating user preferences and risky behaviors for algorithms

Approaches	*Utility*	*Cost*	*Pref*	*Safe*
GLAD (ours)	**-2415.7**	-2265.7	**0.0**	**-150.0**
TMPUD	-2569.2	-1936.2	-333.0	-300.0
TPAD	-2834.0	**-1334.0**	**0.0**	-1500.0
MINI	-2993.0	-2543.0	-300.0	**-150.0**

4.6.2 Evaluation Metrics and Three Baselines

The main measure for evaluation is utility, which considers traveling cost, driving safety, and user preference. The cost was evaluated using traveling distance in meter. Each collision or unsafe behavior incurred a penalty cost of 15000.0. The violation cost in $Pref()$ is 300.0.

In our experiments, we utilized three baselines. The first is **TPAD**, which enforces all task-level actions to be executed by the vehicle at the motion level while driving behaviors'

safety values are ignored [51]. The second baseline is **TMPUD**, which considers motion costs and safety in task-motion planning, but does not model user preferences or visual perception [13]. The final, and notably the weakest, baseline is **MINI**. It considers safety and user preferences, and minimizes plan length, but it does not compute motion costs.

We use GLAD to refer to the default implementation whose safety estimator was realized using ANN (see Section 4.5). By comparison, we use GLAD-SVM to refer to the version that used SVM (for replacing ANN). The safety estimator implemented by ANN is the default denoted as SE-ANN, and the other is denoted as SE-SVM.

4.6.3 Full Simulation

Experiments conducted in CARLA are referred to as being in **full simulation**. We first spawned $\sigma = 120$ vehicles with random models and colors in random positions on the map. All of them were in autopilot mode, where their speed is less than $30 km/h$. Then our ego vehicle was spawned at the initial location, as shown in Figure 4.2. Each reported value corresponds to an average of 100 trials, where on average, each trial took more than 10 minutes to complete. Experiments were performed using a desktop machine (i7 CPU, 32GB memory, and GeForce RTX 3070 GPU).

Results: Table 4.1 summarizes the performances of GLAD (ours) and three baselines. GLAD achieved the highest overall utility, while at the same time performing the best in respecting user preferences ("*Pref*") and maintaining safety ("*Safe*"). In "*Cost*," baseline TPAD produced the best performance, because it does not evaluate execution-time safety and frequently runs into accidents.

Table 4.2: Performance of safety estimators SE-ANN and SE-SVM under two training settings: Default and Non-default (#)

Pct. of D_train	**SE-ANN**	**SE-SVM**
33%	79.1	77.8
66%	83.8	81.5
100%	84.7	82.3
	SE-ANN(#)	**SE-SVM(#)**
33%	79.5	76.7
66%	80.0	79.1
100%	81.7	79.7

Results from Safety Estimator: Table. 4.2 shows the performances of SE-ANN and SE-SVM in F1-score under two settings (default and non-default). From the results, we observe that SE-ANN consistently performed better than SE-SVM. Another observation is that safety estimation is more difficult under the non-default setting, where the vehicle used different roads for training and testing. Finally, we could see that the performances of both methods increase with more training data.

4.6.4 Abstract Simulation

It is computationally expensive to run experiments in full simulation with CARLA, e.g., running the experiments reported in Section 4.6.3 took more than 72 hours on a modern desktop machine. For extensive evaluations, we developed an **abstract simulation** platform, where the perception component is abstracted. Based on the performance reported in Table. 4.2, we modeled the vision-based safety estimator using a 2-by-2 confusion matrix. In the simulation, we first sampled from the confusion matrix, and then accordingly sampled a safety value μ based on our safety estimator's performance on our dataset. Further, we created two traffic conditions in abstract simulation, heavy traffic and normal traffic,

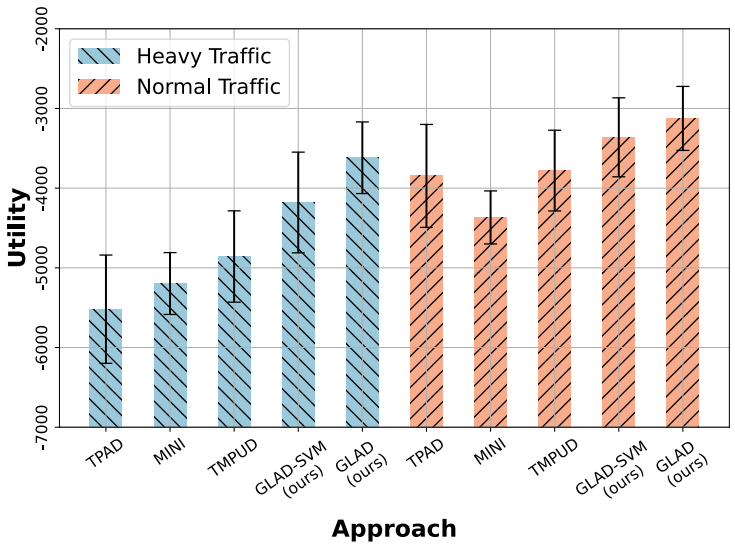


Figure 4.4: Overall performances of GLAD and baselines, where the *x-axis* represents different methods, and the *y-axis* denotes the average utility value. We also reported the standard deviations on top of each bar.

that are associated with different λ values, where λ is the probability of collisions (or unsafe behaviors). In experiments, $\lambda = 0.05$ (or 0.08) under normal (or heavy) traffic. Each data point in abstract simulation corresponds to an average of 6400 trials, where we also evaluated their standard deviations.

Results from Abstract Simulation: Figure 4.4 shows the overall performances of GLAD and the baselines under normal and heavy traffic conditions. We see that GLAD produced the highest utility under both traffic conditions. The experimental results demonstrate that GLAD performed the best in maximizing overall utility, which considers task-completion efficiency, road safety, and user preferences. We also observe that GLAD outperformed the baselines by a larger margin under heavy traffic, because GLAD is good at evaluating execution-time safety to adjust task plans.

5 Planning with Novel Objects

5.1 Introduction

Robots that operate in the real world need to plan at both task and motion levels. At the task level, the robot computes a sequence of symbolic actions in a discrete space to achieve long-term goals [102]. At the motion level, the symbolic actions are implemented in a continuous space, and the computed trajectories can be directly applied to the real world [103]. Planning at task and motion levels at the same time is challenging [104, 105, 91, 93, 106, 95], resulting in the so-called integrated *task and motion planning* (**TAMP**) problem [1, 90], where the main challenge is to achieve task-level goals while maintaining motion-level feasibility.

Planning algorithms (at the task, motion, or both levels) frequently assume that a world model is provided beforehand, including how the world reacts to robot behaviors (i.e., world dynamics). However, many real-world scenarios are very complex, making it hardly possible to model all objects or their physical properties at planning time, e.g., kitchens and shopping centers. In such scenarios, modeling how a robot interacts with multiple objects is even more challenging, e.g., holding a stack of plates, cutting onions, and squeezing through a crowd. Figure 5.1 shows a "**stack-and-push**" task as an example scenario with complex multi-object interactions, where a robot repeatedly builds a block tower and then

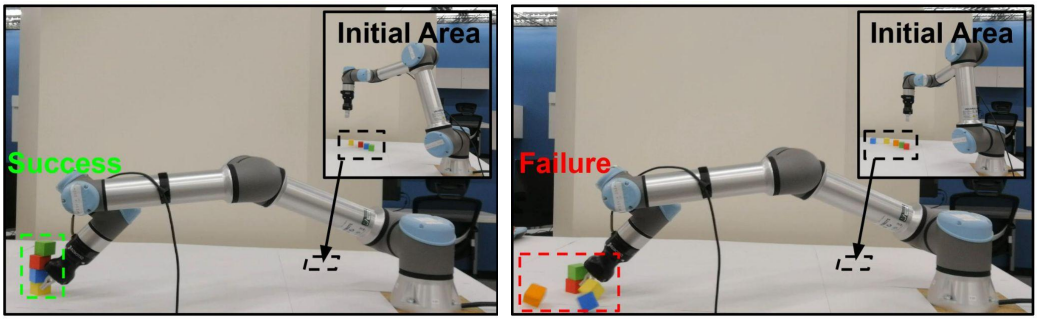


Figure 5.1: A robot performing "stack-and-push" tasks by repeatedly building a stack of blocks and pushing the stack to a goal area. There is a trade-off between stability and efficiency (higher stacks are less stable) that the robot must learn from trial and error. It is a TAMP problem, e.g., the stacking behaviors require both task planning and motion planning.

pushes it to a goal area. Aiming to move all blocks to the goal area, the robot needs to learn object properties (e.g., size and weight), how to grasp the blocks (which depends on the object properties), and a stack-and-push strategy (e.g., how many blocks to be stacked together).

The **first contribution** of this work is a new grounded, object-centric TAMP framework, called task and motion Planning with Physics-based Simulation (**PPS**). The uniqueness of PPS lies in the inclusion of a physics engine for grounding objects and their physical properties. Object grounding enables the robot to collect simulated interaction experiences for policy learning purposes. The PPS framework is general enough to accommodate different TAMP problems that vary in model completeness.

Our **second contribution** is an algorithm, called Task-Motion Object-Centric planning (**TMOC**), that addresses a challenging PPS problem where *object properties are not provided*. TMOC is particularly useful for those TAMP domains that involve complex multi-object dynamic interactions. Our TMOC robot can ground objects in a high-fidelity

physics engine, learn object properties to facilitate the grounding, and improve its task-motion planning skills.

We have demonstrated and evaluated TMOC in simulation (where the robot uses a container to move objects), and using a real robot (where the robot conducts stack-and-push tasks). The simulation and real-world scenarios share the challenge of robot task-motion planning with unknown object properties. From the results, we see that TMOC outperforms a set of baselines from literature in cumulative utility. Finally, we demonstrate the learning process of a real robot that uses TMOC for task-motion planning.

5.2 Related Work

There is a long history of developing planning algorithms in robotics research. We summarize three research areas that are the most relevant to this research, namely TAMP, symbol learning for robot planning, and sim-to-real transfer.

Task and Motion Planning (TAMP): Broadly, any robots that plan behaviors at a high level and operate in the real world would need algorithms for TAMP. However, it is not until recently that TAMP has been used as a term to refer to the algorithms that interleave the processes of task planning and motion planning [1, 90, 95]. Early examples include aSyMov [63] and Semantic Attachment [107]. Hierarchical planning in the now (HPN) [108] has been extended to model the uncertainty in action outcomes and observability [104]. FFRob directly conducts task planning over a set of samples generated in the configuration space [49]. Probabilistically complete TAMP was achieved using constraint satisfaction methods [93]. Wang et al. used a policy synthesis approach to account for uncontrollable

agents (such as humans) [109]. Off-the-shelf task-motion planners can be integrated using a planner-independent interface [47]. Optimization methods have been applied to TAMP domains, where the goal is specified with a cost function [91]. In comparison to planning methods (including TAMP) that assume knowledge of complete world models, this work considers robots that compute world models (e.g., objects' physical properties) through perception, and actively estimate the current world state, which renders "grounding" necessary.

Very recently, researchers developed a visually grounded TAMP approach, called GROP, for mobile manipulation [96]. Compared with GROP that uses computer vision techniques to learn a state mapping function, TMOC (ours) learns to ground objects and their physical properties for TAMP tasks.

Symbol Learning for Robot Planning: Researchers have developed symbol learning methods for robot planning. Konidaris et al. (2018) focused on learning symbols to construct representations that are provably capable of evaluating plans composed of sequences of those actions [110], assuming the robot was equipped with a collection of high-level actions. Their mobile manipulator learned a grounded symbol that indicates that a cupboard door is open, which is necessary for determining whether or not the robot can pick up a bottle from the cupboard. Gopalan et al. (2020) introduced natural language into the loop, and showed that the learned symbols can enable a robot to learn to map natural language to goal-based planning with only trajectories as supervision [111]. The above-mentioned methods assumed that low-level motion primitives are provided (as skills or demonstrations), and the robot needs to learn a symbolic representation that can be used for planning

to accomplish different complex tasks. In this work, we assume those symbols are provided, but there are no models about the symbols' physical meanings in the real world.

Sim-to-Real Transfer: This work requires a high-fidelity simulator, which is related to "sim-to-real" methods to enable agents to learn from simulation for operating in the real world [112]. Focusing on addressing the reality gap, there are at least two families of sim-to-real methods. One intentionally adds different forms of noise into the simulation environment to learn policies that are robust enough to work in the real world [113, 114, 115, 116, 117]; the other actively updates parameters of the simulator toward generating realistic experience for policy learning purposes [118, 119, 120, 121]. This work falls into the second category of methods from the perspective of updating the simulator's parameters using real-world data. The main difference is that our method is placed in the TAMP context. As a result, our proposed method faces two simultaneous, interdependent reality gaps at task and motion levels, respectively.

This work enables robots to plan at both task and motion levels in object-centric domains where the object properties can be unknown. Our developed approach is particularly suitable for task-motion domains that involve complex multi-object dynamic interactions. Next, we define the grounded TAMP problem and then describe our algorithm.

5.3 Framework and Problem Statements

In this section, we first define an object-centric TAMP framework, and then describe how the framework accommodates a few TAMP problems.

5.3.1 Framework

Planning with Physics-based Simulation (**PPS**) is a task-motion planning framework[1], and is defined as tuple

$$\langle Obj, D^t, D^m, Y \rangle,$$

where Obj is a finite set of objects, and the other components are described next.

Task-level Domain Description $D^t : \langle E, A, T \rangle$, where E is a finite set of symbolic (discrete) properties, and $E^{obj} \subseteq E$ includes symbolic properties that are applicable to $obj \in Obj$. Each property $e \in E^{obj}$ has a finite domain of values, referred to as V_e^{obj}. For instance, in a manipulation domain, object $box \in Obj$ has a property of $loaded \in E^{box}$ that has domain $V_{loaded}^{box} = \{true, false\}$. A state s is an assignment of each object's applicable symbolic properties. S denotes the set of possible states in symbolic forms. We use $s^{init} \in S$ and $S^G \subseteq S$ to represent the initial state and the set of goal states, respectively.

A is a finite set of object-centric actions, and action $a \in A$ is defined by its preconditions and effects. State transition model $T(s, a, s')$ defines how an action leads (deterministic or probabilistic) transitions. When action costs are considered in task planning, a task planner is able to select a plan out of all the plans that satisfy the goal conditions toward minimizing the overall cost. In this work, at the task level, we are concerned with object-centric, cost-sensitive, goal-conditioned planning under uncertainty.

Motion-level Domain Description $D^m : \langle L, C \rangle$, where L is a finite set of continuous properties. L is the counterpart of E at the task level. $L^{obj} \subseteq L$ includes properties that are

[1]Researchers have developed TAMP frameworks, such as PETLON for navigation domains [95], and FFRob [49] for heuristics-based TAMP. The uniqueness of PPS, as an object-centric TAMP framework, is attributed to its inclusions of a state mapping function, and a high-fidelity simulator.

applicable to $obj \in Obj$, and properties $l \in L^{obj}$ is a real number. Example continuous properties include an object's size, position, and weight. The configuration space C is the set of all possible configurations. We use ξ to represent a motion trajectory, where the trajectory connects the robot's current configuration (x^{init}) to any configuration in the target space ($x^{goal} \in C$).

State Mapping Function Y maps a task-level state to a motion-level pose: $x \leftarrow Y(s)$, where s is a task-level state and $x \in C$ is a configuration. Y^{-1} is the inverse function of Y, and outputs a symbolic state s given a motion-level configuration. Here we assume Y^{-1} can be derived from Y, and is thus omitted from the definition of PPS.

PPS is a general-purpose TAMP framework that is able to accommodate different task-motion planning problems. A realization of a PPS framework requires a task planning system, a motion planning system, and a physics-based simulation (physics engine) system. One can leverage the physics engine of PPS to ground different components of the real world.

Next, we define a challenging PPS problem, and discuss how this problem connects to a few existing TAMP problems.

5.3.2 Problem Statements

In this work, we aim to address the PPS problem with the following set of functions being unknown:

$$\{L, Y, T\}$$

Consider a "**stack-and-push**" domain: A robot does not know block size and weight

(about L), where to place the gripper for grasping (about Y), and how stable a stack of N blocks is (about T). This is a challenging problem, because planning at task and motion levels highly depends on the estimation of object properties, and the task-level strategy further depends on the robot's motion-level performance. We use the stack-and-push domain in real-robot experiments, where the robot aims to move blocks from an initial area to a goal area as quickly and stably as possible.

The PPS framework is general enough to accommodate different TAMP problems (in addition to the above-mentioned problem that is the focus of this work). For instance, when all components of PPS are known, it corresponds to a standard TAMP problem, e.g., [95, 13]. When only transition function T is unknown, it corresponds to existing research, such as [122, 93]. Planning domains with unknown physical properties L [70], and planning domains with unknown state mapping function Y [123] can be modeled as PPS problems as well.

The input of a PPS algorithm includes a PPS domain in the form of $\langle Obj, D^t, D^m, Y \rangle$, and a set of task-level goal states S^G. Our utility function incorporates action costs, success bonus, and failure penalty. A PPS algorithm aims to compute task-motion plans to achieve task-level goals while maximizing cumulative utilities.

Next, we focus on our PPS problem with unknown object properties L (where the robot is motivated to learn Y and T accordingly), and develop algorithm TMOC to enable robots to learn to plan at task and motion levels.

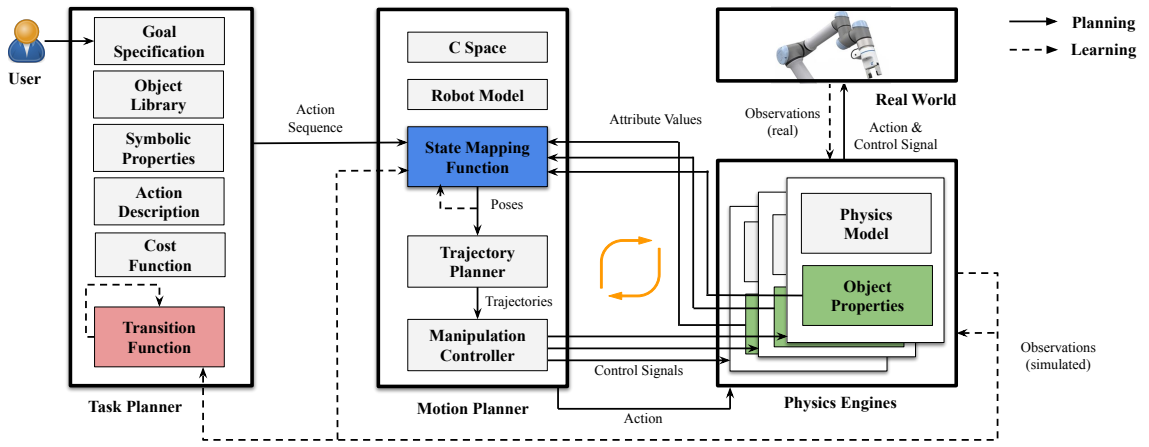


Figure 5.2: An overview of the TMOC algorithm for our PPS problem with unknown object properties (L in green), state mapping function (Y in blue), and transition function (T in red). The main components of TMOC include a task planner for sequencing symbolic actions, a motion planner for computing motion trajectories, and a physics engine for simulating multiple grounded worlds. TMOC takes object library, task domain description, goal specification, cost function, motion domain description and state mapping function as input. TMOC aims to compute task-motion plans to achieve task-level goals while maximizing cumulative utilities. Using TMOC, a robot learns object properties (L) from task-completion experience, and leverages the learned properties to further improve its task-motion planning skills (Y and T) over time.

5.4 Algorithm

In this section, we present the main contribution of this research, **TMOC** (short for "task-motion object-centric" planning), a grounded task-motion planning algorithm for addressing the PPS problem described in Section 5.3.2, where object properties (L), state mapping function (Y), and transition function (T) are unknown. TMOC includes three main components of a task planner, a motion planner, and a physics engine, as illustrated in Figure 5.2.

Algorithm 4 presents the control loops of TMOC. The input of TMOC includes a PPS domain in the form of $\langle Obj, D^t, D^m, Y \rangle$, and a set of task-level goal states S^G. TMOC learns

L, Y, and T in each iteration, where the robot completes a task *once* in the real world and *N times* in simulation. In each iteration, each of L, Y, and T is learned under the current estimation of the other two, while a TMOC agent plans at task and motion levels.

5.4.1 TMOC algorithm

A realization of a PPS framework requires task planning system P^t, motion planning system P^m, and physics-based simulation system $\mathscr{M}$. P^t takes task domain D^t together with the current state s (which can be derived from D^t) as input and generates an action sequence. Specifically, P^m takes the initial configuration x^{init}, goal configuration set X^G, and D^t as input, and generates motion trajectory ξ. $\mathscr{M}$ takes ξ, object set Obj, and their physics-relevant properties L as input, and generates L' denoting the resulting properties of Obj. From L', one can infer motion planning domain D^m, which can be further used for computing symbolic state s using the function Y^{-1}.

Structure of Algorithm TMOC: Lines 1∼3 are for initializing data structures. Lines 5∼24 form a complete iteration. In each iteration, Line 5 computes a task-level plan, and the functions of Y, L, and T are updated from trial and error in Lines 8, 22, and 23, respectively. Lines 7∼14 are for executing an action in the real world; and Lines 15∼21 are for executing the same action in simulation. TMOC is a life-long learning algorithm, and does not have termination conditions. Next, we look into individual lines of TMOC.

Line 1 initializes transition function T "optimistically". A transition function (T) indicates how reliable an action is. Initializing T optimistically means that the robot believes it always gets the desired results after performing an action, e.g., manipulation and push actions are always successful. This initialization strategy is inspired by the R-max algo-

Algorithm 4 TMOC algorithm

Require: P^t, P^m, and $\mathscr{M}$
Input: $\langle Obj, D^t, D^m, Y \rangle$, S^G and $Cost$

1: Initialize T optimistically
2: Initialize N simulated worlds and each one has L_i (particle) with a uniform weight w_i, where $i = 0, 1, \cdots, N-1$
3: Initialize an empty experience pool *Pool*
4: **while** True **do** ▷ start working on a new trial
5: $p \leftarrow P^t(\mathscr{D}^t, s^{init}, S^G, Cost)$, where D^t includes T ▷ this is task planning
6: **for** each action $\langle s, a \rangle$ in p **do**
7: **for** $i = 0, 1, \cdots, N-1$ **do**
8: Compute Y_i using L_i for its simulated world
9: Generate a feasible pose x_i for $\langle s, a \rangle$ using Y_i
10: **end for**
11: Compute a pose x for the real world by weighted averaging poses $\{x_0, x_1, \cdots, x_{N-1}\}$
12: $\xi \leftarrow P^m(x_{curr}, x, D^m)$ ▷ this is motion planning
13: Execute ξ in the real world and then obtain a resulting state s'
14: $Pool = Pool \cup \{\langle s, a, s' \rangle\}$
15: **for** $i = 0, 1, \cdots, N-1$ **do**
16: $\xi_i \leftarrow P^m(x_{curr}, x_i, D^m)$
17: $L_i' \leftarrow \mathscr{M}(\xi_i, Obj, L_i)$, where L_i is in D^m
18: Infer a resulting state s_i' using L_i' and Y_i^{-1}, and then $L_i \leftarrow L_i'$
19: $Pool = Pool \cup \{\langle s, a, s_i' \rangle\}$
20: Compute weight w_i of L_i using s' and s_i'
21: **end for**
22: Update L_i from the discrete distribution given by $\{w_0, w_1, \cdots, w_{N-1}\}$
23: Update T using *Pool*
24: **end for**
25: **end while**

rithm [124], and encourages exploration in the early learning phase. TMOC initializes N simulated worlds, where each is specified by a set of physics-relevant properties L_i, e.g., length, width, and density of blocks (Line 2). Each simulated world is associated with a weight w_i, and the weights are uniformly initialized. An empty experience pool *Pool* is initialized to store observations (i.e., actions and their resulting states) after action executions (Line 3).

Line 5 computes an optimal action sequence p using the task planning system P^t given

the utility function, which takes into account transition function T (e.g., how reliable grasps are) and cost function *Cost* (e.g., how long it takes a robot to grasp an object).

Lines 7∼14 describe how action $\langle s,a\rangle$ is implemented in the **real world**. From L_i, we can get the configuration space, based on which Y_i can be computed using motion planning methods (Line 8). Thus, a feasible pose x_i for action $\langle s,a\rangle$ can be generated using Y_i (Line 9). A new pose x for the real world is computed by weighted averaging poses x_i, where $i = 0,1,\cdots,N-1$ (Line 11). A trajectory ξ is computed using the motion planner P^m, which takes x_{curr}, computed x, and D^m as input, where x_{curr} refers to the current pose of the robot (Line 12). ξ is executed in the real world, and thus a resulting state s' of action $\langle s,a\rangle$ is obtained (Line 13). Finally, tuple $\langle s,a,s'\rangle$ is added to experience pool *Pool* (Line 14).

Lines 15∼21 explain how action $\langle s,a\rangle$ is implemented in all **simulated worlds** (one iteration for each world). A trajectory ξ_i is computed using motion planning system P^m (Line 16). Then, a set of resulting properties L_i' is obtained using physical-based simulation system $\mathscr{M}$ (Line 17). Thus, a resulting state s_i' can be inferred using L_i' and the reverse function of Y_i, and L_i is also updated by L_i' (Line 18). Finally, tuple $\langle s,a,s_i'\rangle$ is added to *Pool* (Line 19).

Lines 20, 22 $\sim$ 23 update L and T using action completion experience. Specifically, each weight w_i of L_i can be computed based on the resulting state from the real and simulated worlds (Line 20). Similar to the particle filter [125], a new L_i is sampled from the discrete distribution given by w_i, where $i = 0,1,\cdots,N-1$ (Line 22). Besides, T is updated using the experience pool *Pool* (Line 23).

TMOC enables robots to plan at both task and motion levels in object-centric domains

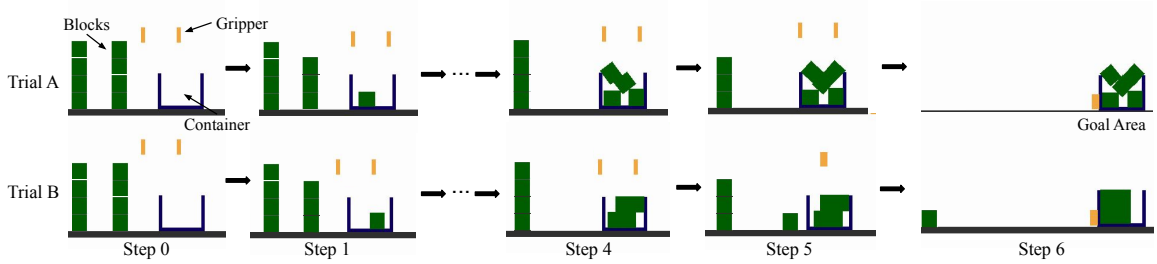


Figure 5.3: A robot needs to stack a set of blocks (Steps $0 \sim 6$ of Trial A and B) and then push them to the goal area with a container. The more blocks in the container, the less stable it is, resulting in a trade-off between work efficiency and success rate. At the task level, the robot is concerned about how many blocks (of initially unknown sizes and weights) can be placed in the container. Step 4 of trials A and B demonstrate two such situations, where the robot needs to decide if another block can be added to the container. Making this decision is difficult due to the complex multi-object interactions. The two trials give different results as shown in Step 5 of trials A and B.

where the objects' physical properties (L) are unknown (and thus T and Y are unknown). TMOC is particularly useful for those tasks that involve dynamic complex robot-multi-object interactions that can hardly be modeled beforehand. Next, we focus on the evaluation of TMOC in both simulation and the real world.

5.4.2 Algorithm Instantiation

Task Planner: Our task planner P^t is implemented using Answer Set Programming (ASP), which is a popular declarative language for knowledge representation and reasoning. ASP has been applied to task planning [20, 69, 95, 70]. In out domain, predicate `is_holding(R1,B1)` is used to specify block `B1` being in the robot hand `R1`. We model three manipulation actions, including `pickup`, `stack`, and `push`. For instance, action `pickup` is used to help the robot arm pick up the target block from an initial location, where constraints, such as "`stack` is allowed only if a target block is in the robot hand", have been modeled as well.

Table 5.1: Action knowledge in our system, organized into preconditions and effects.

Action	Precondition	Effect
$pickup(R,B,L)$	$is\_in(B,L)$ $hand\_empty(R)$	$is\_holding(R,B)$ $\neg hand\_empty(R)$
$stack(R,B,C)$	$is\_holding(R,B)$	$\neg is\_holding(R,B)$ $hand\_empty(R)$ $at\_container(B,C)$
$push(R,C,L1,L2)$	$hand\_empty(R)$ $is\_in(C,L1)$	$hand\_empty(R)$ $is\_in(C,L2)$

An example goal specification can be "all blocks are in the container, and containers are at the goal location". Table 5.1 defines three actions by their preconditions and effects, where R, B, C, and L stand for robot, block, container, and location, respectively.

Motion Planner: At the motion level, given a configuration space, a roadmap is firstly built for the robot. A trajectory planner then generates a desired continuous and collision-free trajectory with minimal traveling distance using optimization algorithms, RRT* in our case [126]. The trajectory includes a set of poses, and the trajectory is delivered to the manipulation controller, along with the robot's current pose.

5.5 Experiments

We have conducted experiments both in simulation and using a real robot. In simulation experiments, we focus on statistically comparing the performances of TMOC and a set of competitive baselines using results collected from large numbers of trials. The comparisons are based on cumulative utility, which is a combined measurement that incorporates action costs, success bonus, and failure penalty.[2] In real-world experiments, we illustrate a

[2]The action cost of *pickup*, *stack*, and *push* are 15, 10 and 30, respectively, where a cost technically corresponds to a negative reward. The bonus of successfully stacking a block, and pushing a container to a

complete learning process of a real robot arm performing "stack and push" tasks.

5.5.1 Experiments in Simulation

We used an open-source 2D physics-based simulator called Pybox2D [127] to evaluate the performance of TMOC. We decided to use a 2D simulator instead of a 3D one because 3D simulators (e.g., Gazebo [39] and PyBullet [128]) are generally computationally less efficient. Our agent needs to learn to interact with objects (with unknown properties) at both task and motion levels, which requires considerable interaction experience, so we selected a 2D simulator that allows running experiments extensively.

Figure 5.3 illustrates how a robot gripper uses a container to move eight blocks to a goal area[3]. The robot knows that all blocks share the same density, but must estimate their sizes for task-motion planning. The robot might fail in loading and unloading a block, and in blocks falling off from the container. Here we assume a "helper" helps move blocks out of the container in the goal area before the gripper and container are moved back. We use this task to capture complex contact-based interactions among multiple objects, whereas the interactions in "stack-and-push" scenarios (used for real-robot experiments) do not go beyond two objects.

Baselines: Three baseline methods are utilized in this research, and they are selected from the literature, referred to as **TMP-RL** [70], **GP** [122], and **TOEP** [123].

- TMP-RL is a task-motion planning algorithm that learns from trial and error. TMP-RL is not object-centric, and hence does not learn object properties (L) over time.

goal location is 40 and 80, respectively. The failure penalty is 50.

[3] In this work, we assume blocks and the container are not too heavy to be manipulated by the robot. Besides, the blocks are small enough to be graspable by the robot arm.

- GP is a task-motion planning algorithm that learns primitive skills from trial and error, but cannot improve its task-level planning strategies. In our implementation of GP, the agent randomly selects one of the satisficing plans to achieve high-level goals.

- TOEP does not learn the state mapping function, and the gripper is placed (on a block) in a grasping position that is randomly selected in a reasonable range. It should be noted that our "TOEP" agent is equipped with task-planning capability, whereas the original TOEP work used predefined task-level behaviors.

TMOC vs. Three Baselines: For every approach, we conducted 15 runs with 5000 episodes in each run. Figure 5.4 (Top Left) illustrates the learning curves for utility value, averaged over the 15 runs with the shaded regions representing one standard deviation from the mean. From the results, we get the important observation that TMOC [4] performs better than the baselines in terms of cumulative utility and learning rate.

One interesting observation from Figure 5.4 (Top Left) is that TOEP fell behind the other three approaches (including our TMOC) at the beginning, and then later reached a cumulative utility level that is comparable to that of TMOC. This is because TOEP does not learn the state mapping function, and hence cannot improve its skills of "grasping" and "pushing" from trial and error. This disadvantage affects the learning rate of TOEP at the beginning. At the late learning phase (after about 2000 episodes), the TOEP agent was able to catch up, because the task planner learned to adapt to the motion planner (that is suboptimal on grasping and pushing). As a result, the ultimate performances of TOEP

[4] The number of grounded worlds in TMOC is 200 (i.e., $N = 200$).

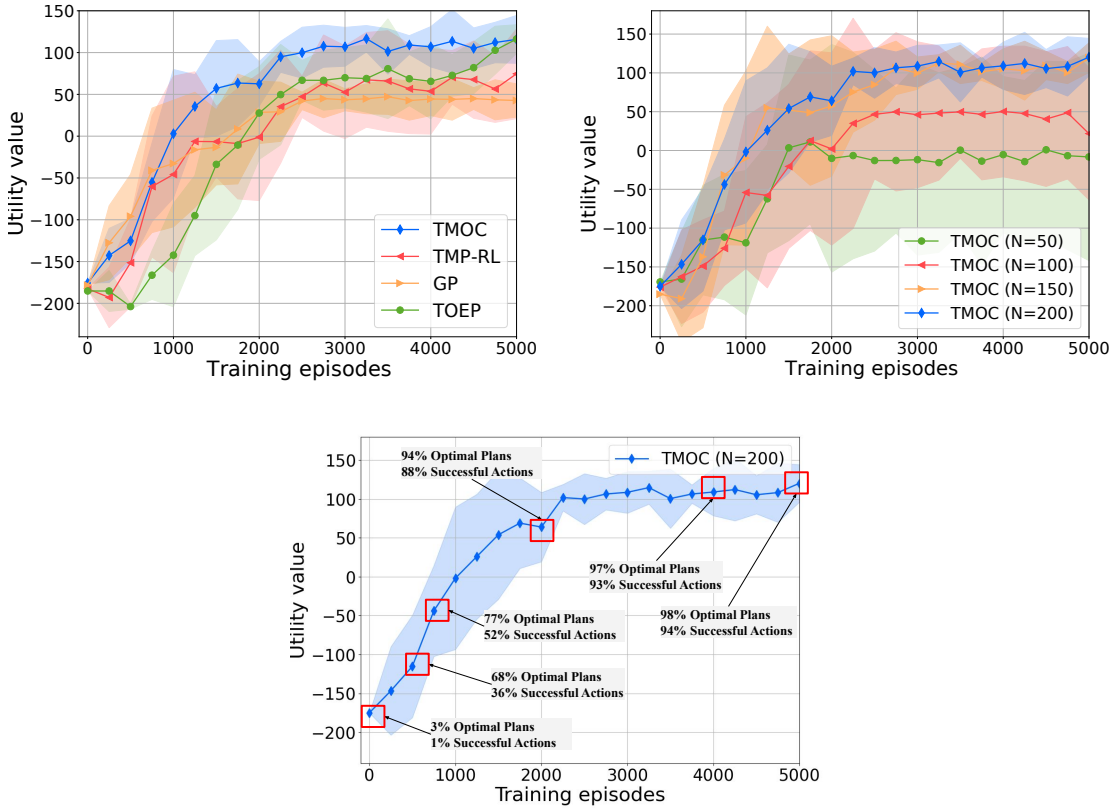


Figure 5.4: **Top Left**: Comparisons between TMOC and three baselines of TMP-RL, GP and TOEP in the task of a gripper moving objects using a container; **Top Right**: Performance of TMOC under different numbers of simulated worlds, where N = 50, 100, 150, and 200; **Bottom**: Milestones of TMOC, where we present the percentages of optimal plans and successful actions. In all subfigures, the y-axis refers to the cumulative utility obtained by the real robot.

and TMOC (ours) are comparable. TMP-RL does not learn object properties (L), which affected its learning rate and cumulative utility level compared with TMOC (ours), because an inaccurate L is detrimental to the learning of both Y and T.

Number of Grounded Worlds: The performance of TMOC highly depends on the number of simulated worlds (or "particles," in the terminology of Particle Filters), which is referred to in Line 2 of Algorithm 4. Each curve was based on 12 runs with 5000 episodes in each run. Sufficient particles are necessary to ensure the quality of a Particle filter, and to

represent the distributions being estimated. We are interested in answering this question: *How many particles are adequate for our experiments?* Thus, we evaluated TMOC under different numbers of particles (i.e., $50 \leq N \leq 200$).

Figure 5.4 (Top Right) shows no significant improvement in learning rate or cumulative utility when we increased the particle number from $N = 150$ to $N = 200$. Therefore, we believe $N = 150$ is sufficient to the robot in our domain. This observation can serve as a reference to TMOC practitioners.

Milestones of TMOC: We are also interested in how the robot makes progress in learning L, Y, and T while running TMOC. Thus, we calculated the percentages of optimal plans and successful actions at some selected training episodes, where $N = 200$ in this experiment.

Figure 5.4 (Bottom) shows that both percentages of optimal plans and successful actions are close to 0% at the beginning, because of the highly inaccurate functions of L, Y, and T. They gradually increased and finally got close to 100% after 5000 episodes. At the same time, we see that L, Y, and T have different learning rates. For instance, when the robot completed about 2000 episodes, T converged well because most of the plans (about 94%) are optimal, while there is still room to further learn functions L and Y because only 88% of actions are successful. We attribute the growth of utility value after 2000 episodes to the learning of L and Y.

Task Variations of TMOC: The performance of TMOC is affected by task variations, such as blocks of different sizes. Thus, we changed the goal specification to let the robot stack and move one big block and seven small ones. The side length of the big one is 20% bigger than the small ones. The robot knows that all blocks share the same density, but

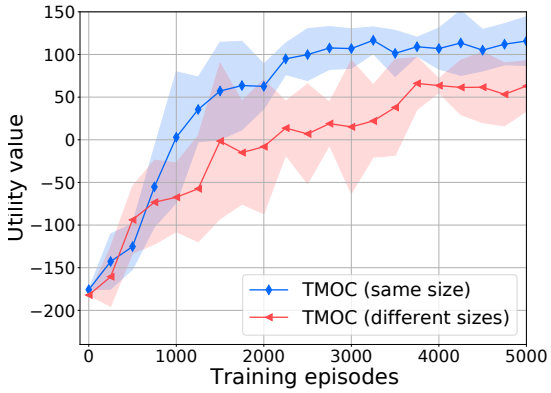


Figure 5.5: Performance of TMOC under a task variation (i.e., size). **Same Size**: the robot is tasked to stack and move eight small blocks. **Different Sizes**: the robot is tasked to stack and move one big block and seven small ones. In both curves, N equals 200.

Table 5.2: Utility value of TMOC under different episodes in the stack-and-push task using real robot arm UR5e

Episode	0	4	8	12	16
Utility	−23(18)	60(128)	−10(32)	4(67)	183(80)
Episode	20	24	28	32	36
Utility	70(98)	86(113)	105(74)	218(11)	235(0)

must estimate their sizes for task-motion planning. Introducing different sizes makes the stack-and-push task too difficult for the robot. For evaluation purposes, we provided the robot with guidance that the big block should be put near the bottom (in one of the first two steps).

Figure 5.5 shows that under the "different sizes" domain variation, TMOC needs more episodes to reach a utility level that was achieved when all blocks share the same shape. Also, under the "different sizes" variation, TMOC's convergence level was lower than that under the "same size" variation. We observed that grasps become more unreliable when the robot faces a bigger block. Incidentally, we found no significant difference in the performance of TMOC with and without block weight variations.

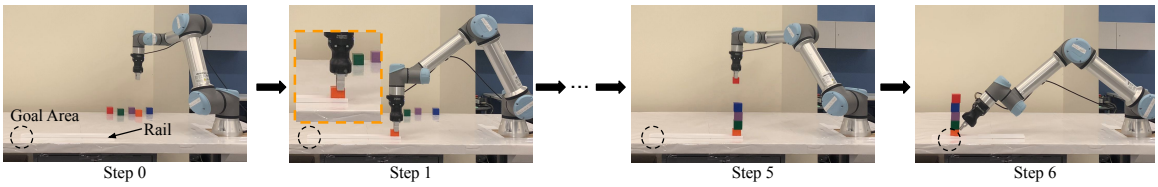


Figure 5.6: A robot arm is tasked to move five blocks from an initial area. To perform "2D" experiments in the 3D real world, we attached a rail onto a table, where blocks can be only moved back and forth. There is a trade-off between stability and efficiency (higher stacks are less stable) that the robot must learn from trial and error. In this case, five blocks are stacked and then moved to the goal area. The robot was lucky enough to be successful in this trial, but might fail in other trials of performing the same plan.

5.5.2 Experiments in the Real World

TMOC was evaluated using a UR5e robotic arm. In the experiment setting, the robot stacked blocks and then pushed them to a goal area, as illustrated in Figure 5.6. We provided the robot with accurate object properties (block size in this case) to control the difficulty of learning task-motion behaviors to a reasonable level. As a result, the robot only needed to learn a state mapping function (e.g., where to place the gripper for grasping) and a transition function (e.g., a stack-and-push strategy, including how many blocks should be stacked together). We conducted three runs, each including 44 episodes, that together took more than 12 hours.

Table 5.2 shows the mean utility values and the corresponding standard variances of the TMOC algorithm after different numbers of episodes. At the beginning, the mean value is low while the variance is high, because the robot does not know how to grasp blocks or what a good action sequence is like. With more trials, our TMOC robot learned a good state mapping function Y and transition function T, which resulted in a high utility value and a low variance. More specifically, the robot learned reasonably good Y and T

functions after 15 and 32 episodes, respectively. The result demonstrates that learning Y and T enables the robot to complete the stack-and-push task efficiently and reliably, which is overall evaluated using the utility values.

6 Planning to Fulfill Underspecified Requests

6.1 Introduction

Multi-object rearrangement is a critical skill for service robots to complete everyday tasks, such as setting tables, organizing bookshelves, and loading dishwashers [129, 130]. These tasks demand robots exhibit both manipulation and navigation capabilities. For example, a robot tasked with setting a dinner table might need to retrieve tableware objects like a fork or a knife from different locations and place them onto a table surrounded by chairs, as shown in Figure 6.1. To complete the task, the robot needs to correctly position the tableware objects in semantically meaningful configurations (e.g., a fork is typically on the left of a knife) and efficiently navigate indoors while avoiding obstacles like chairs or humans whose locations are unknown in advance.

A variety of mobile manipulation systems have been developed for object rearrangement tasks [131, 132, 133, 134, 135, 136, 137, 138]. Most of those systems require explicit instructions, such as arranging similar colored items in a line or placing them in a specific shape on a table [131, 134, 135, 137, 138, 139]. However, user requests in the real world tend to be underspecified: there can be many different ways to set a table that are not equally preferred. How does a robot figure out a fork should be placed on the left of a plate and a knife on the right? Considerable commonsense knowledge is needed. Recent results have

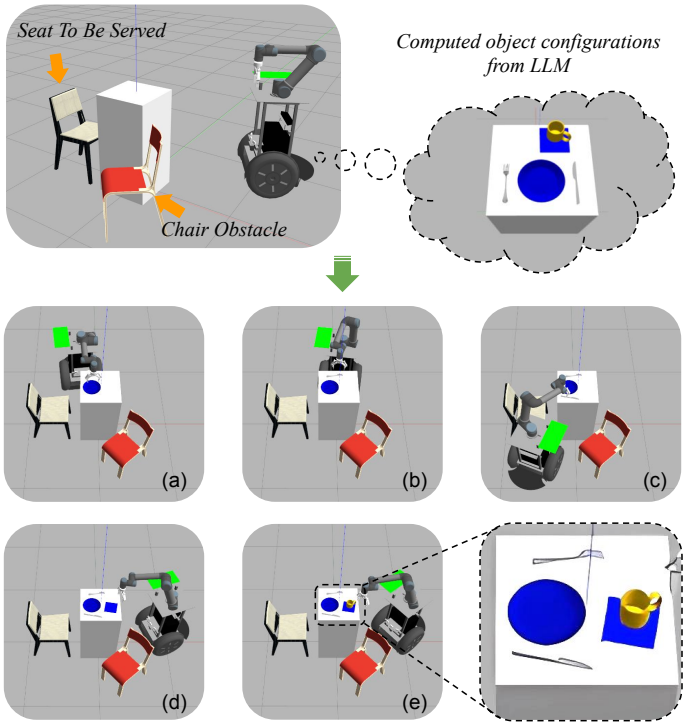


Figure 6.1: A mobile manipulator is assigned the task of setting a table in a dining domain. The manipulator needs to arrange several tableware objects, including a knife, a fork, a plate, a cup mat, and a mug. These objects are available on the other tables, and there are also randomly generated obstacles (i.e., the red chair) that are not included in the pre-built map beforehand. The robot needs to compute feasible and efficient plans for rearranging the objects on the target table using both navigation and manipulation behaviors.

shown large language models (LLMs) like GPT3 [140] and ChatGPT [14] capture a great deal of this common sense knowledge [141]. In the past, researchers have equipped mobile manipulators with semantic information using machine learning methods [132, 142, 133]. Those methods require collecting training data, which limits their applicability to robots working on complex service tasks.

To equip robot planning methods with common sense for object rearrangement, we introduce LLM-GROP, standing for **L**arge **L**anguage **M**odel for **G**rounded **RO**bot Task and Motion **P**lanning, our approach that leverages commonsense knowledge for planning to complete object rearrangement tasks. LLM-GROP first uses an LLM to generate *symbolic* spatial relationships between objects, e.g., a fork and a knife are placed on the left and right respectively. The spatial relationships then can be grounded to different *geometric* spatial relationships whose feasibility levels are evaluated by a motion planning system, e.g., placing objects in some areas of a table can be easier than the others. Finally, the feasibility and efficiency of different task-motion plans are optimized towards maximizing long-term utility, i.e., seeking the best trade-off between motion feasibility and task-completion efficiency.

We have applied LLM-GROP to a dining room, where a mobile manipulator must set a table according to a user's instructions. A set of tableware objects are provided to the robot, where the robot's task is to compute a tabletop configuration of those objects that comply with common sense, and compute a task-motion plan to realize the configuration. To evaluate the performance of our approach, we had users rate different place settings to get a subjective evaluation. We observed improvements in user satisfaction from LLM-GROP compared with existing object rearrangement methods, while maintaining similar

or lower cumulative action costs. Finally, LLM-GROP was demonstrated on a real robot.

6.2 Related Work

We first introduce the object rearrangement domain, then discuss methods for tabletop object arrangement that mostly rely on supervised learning methods, and finally summarize research on using large language models for planning.

Object Rearrangement: Rearranging objects is a critical task for service robots, and much research has focused on moving objects from one location to another and placing them in a new position. Examples include the Habitat Rearrangement Challenge [129] and the AI2-THOR Rearrangement Challenge [130]. There is rich literature on object rearrangement in robotics [131, 134, 135, 137, 138, 139, 96]. A common assumption in those methods is that a goal arrangement is part of the input, and the robot knows the exact desired positions of objects. ALFRED [143] proposed a language-based multi-step object rearrangement task, for which a number of solutions have been proposed that combine high-level skills [144, 145], and which have recently been extended to use LLMs as input [146]. However, these operate at a very coarse, discrete level, instead of making motion-level and placement decisions, and thus can't make granular decisions about common-sense object arrangements. By contrast, our work accepts underspecified instructions from humans, such as setting a dinner table with a few provided tableware objects. LLM-GROP has the capability to do common sense object rearrangement by extracting knowledge from LLMs, and operates both on a high level and on making motion-level placement decisions.

Predicting Complex Object Arrangements: Object arrangement is a task that involves

arranging items on a tabletop to achieve a specific functional, semantically valid goal configuration. This task requires not only the calculation of object positions but also adherence to common sense, such as placing forks to the left and knives to the right when setting a table. Previous studies in this area, such as [132, 147, 142, 133], focused on predicting complex object arrangements based on vague instructions. For instance, StructFormer [148] is a transformer-based neural network for arranging objects into semantically meaningful structures based on natural-language instructions. By comparison, our approach LLM-GROP utilizes an LLM for commonsense acquisition to avoid the need of demonstration data for computing object positions. Additionally, we optimize the feasibility and efficiency of plans for placing tableware objects.

There is recent research for predicting complex object arrangement using web-scale diffusion models [147]. Their approach, called DALL-E-Bot, enables a robot to generate images based on a text description using DALL-E [149], and accordingly arrange objects in a tabletop scenario. Similar to DALL-E-Bot, LLM-GROP achieves zero-shot performance using pre-trained models, but it is not restricted to a single top-down view of a table. In addition, we consider the uncertainty in manipulation and navigation, and optimize efficiency and feasibility in planning.

Robot Planning with Large Language Models: Many LLMs have been developed in recent years, such as BERT [25], GPT-3 [140], ChatGPT [14], CodeX [150], and OPT [151]. These LLMs can encode a large amount of common sense [141] and have been applied to robot task planning [152, 28, 155, 30, 156, 5, 157, 35, 142, 153, 154]. For instance, the work of Huang et. al. showed that LLMs can be used for task planning in household

domains by iteratively augmenting prompts [28]. SayCan is another approach that enabled robot planning with affordance functions to account for action feasibility, where the service requests are specified in natural language (e.g., "make breakfast") [155]. Compared with those methods, LLM-GROP optimizes both feasibility and efficiency while computing semantically valid geometric configurations.

6.3 The LLM-GROP Approach

The objective of this task is to rearrange multiple tableware objects, which are initially scattered at different locations, into a tabletop configuration that is semantically valid and aligns with common sense. The robot is provided with prior knowledge about table shapes and locations, and equipped with skills of loading and unloading tableware objects. There are dynamic obstacles, e.g., chairs around tables, that can only be sensed at planning time. We consider uncertainty in navigation and manipulation behaviors. For instance, the robot can fail in navigation (at planning or execution time) when its goal is too close to tables or chairs, and it can fail in manipulation when it is not close enough to the target position. Note that uncertainties are treated as black boxes in this work.

In this research, we develop LLM-GROP that leverages LLMs to facilitate a mobile manipulator completing object rearrangement tasks. LLM-GROP consists of two key components, LLM for generating symbolic spatial relationships (Sec. 6.3.1) and geometric spatial relationships (Sec. 6.3.2) between objects, and TAMP for computing optimal task-motion plan (Sec. 6.3.3), as shown in Figure 6.2.

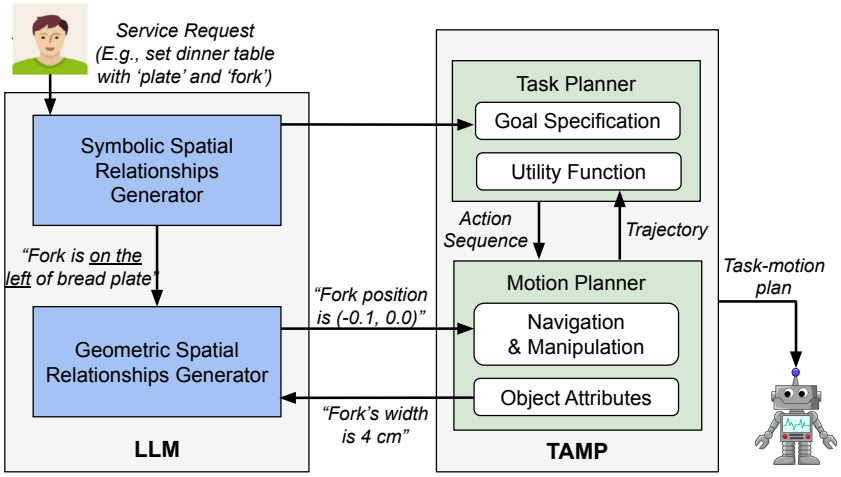


Figure 6.2: LLM-GROP takes service requests from humans for setting tables and produces a task-motion plan that the robot can execute. LLM-GROP is comprised of two key components: the LLM and the Task and Motion Planner. The LLM is responsible for creating both symbolic and geometric spatial relationships between the tableware objects. This provides the necessary context for the robot to understand how the objects should be arranged on the table. The Task and Motion Planner generates the optimal plan for the robot to execute based on the information provided by the LLM.

6.3.1 Generating Symbolic Spatial Relationships

LLMs are used to extract common sense knowledge regarding symbolic spatial relationships among objects placed on a table. This is accomplished through the utilization of a template-based prompt:

> Template 1: *The goal is to set a dining table with objects. The symbolic spatial relationship between objects includes [spatial relationships]. [examples]. What is a typical way of positioning [objects] on a table? [notes].*

where *[spatial relationships]* includes a few spatial relationships, such as *to the left of* and *on top of*. In presence of *[examples]*, the prompting becomes few-shot; when no examples are provided, it is simplified to zero-shot prompting. In practice, few-shot prompts can

ensure that the LLM's response follows a predefined format, though more prompt engineering efforts are needed. *[objects]* refers to the objects to be placed on the table, such as *a plate*, *a fork*, and *knife*. To control the LLM's output, *[notes]* can be added, such as the example *"Each action should be on a separate line starting with 'Place'. The answer cannot include other objects"*.

LLMs are generally reliable in demonstrating common sense, but there may be times when they produce contradictory results. To prevent **logical errors**, a logical reasoning-based approach has been developed to evaluate the consistency of generated candidates with explicit symbolic constraints. This approach is implemented on answer set programming (ASP), which is a declarative programming language that expresses a problem as a set of logical rules and constraints [158]. In the event of a logical inconsistency, the same template is repeatedly fed to the LLM in an attempt to elicit a different, logically consistent output. ASP enables recursive reasoning, where rules and constraints can be defined in terms of other rules and constraints, providing a modular approach to problem-solving [70]. ASP is particularly useful for determining whether sets of rules and constraints are true or false in a given context.

The approach involves defining spatial relationships, their transitions, and rules for detecting conflicts. These rules are created by human experts and serve to ensure that the generated context is logical and feasible. One such rule is `:- below(X,Y),right(X,Y)`, which states that object `X` cannot be both "below" and "to the right of" object `Y` at the same time. This rule ensures that the resulting arrangement of objects is physically possible. An instance of identifying a logical error is provided. For example, an LLM may generate instructions for arranging objects as follows:

1. Place fruit bowl in the center of table.
2. *Place butter knife above and to the right of fruit bowl.*
3. *Place dinner fork to the left of butter knife.*
4. Place dinner knife to the right of butter knife.
5. *Place fruit bowl to the right of dinner fork.*
6. Place water cup below and to the left of dinner knife.

There are **logical inconsistencies** in the italic lines: Steps 2 and 3 suggest placing the *fruit bowl* below the *dinner fork*, while Step 5 suggests placing the *fruit bowl* to the right of the *dinner fork*. This contradicts the established rule and results in no feasible solutions.

6.3.2 Generating Geometric Spatial Relationships

After determining the symbolic spatial relationships between objects in Sec. 6.3.1, we move on to generate their geometric configurations, where we use the following LLM template.

> Template 2: *[object A] is placed [spatial relationship] [object B]. How many centimeters [spatial relationship] [object B] should [object A] be placed?*

For instance, when we use Template 2 to generate prompt "*A dinner plate is placed to the left of a knife. How many centimeters to the left of the water cup should the bread plate be placed?*", GPT-3 produces the output "*Generally, the dinner knife should be placed about 5-7 centimeters to the right of the dinner plate.*"

To determine the positions of objects, we first choose a coordinate origin. This origin could be an object that has a clear spatial relationship to the tabletop and is located centrally.

A dinner plate is a good example of such an object. We then use the recommended distances and the spatial relationships between the objects to determine the coordinates of the other objects. Specifically, we can calculate the coordinates of an object by adding or subtracting the recommended distances in the horizontal and vertical directions, respectively, from the coordinates of the coordinate origin. The LLM-guided position for the ith object is denoted as (x^i, y^i), where $i \in N$.

However, relying solely on the response of the LLMs is not practical as they do not account for object attributes such as shape and size, including tables constraints. To address this limitation, we have designed an adaptive sampling-based method that incorporates object attributes after obtaining the recommended object positions. Specifically, our approach involves sequencing the sampling of each object's position using a 2D Gaussian sampling technique [159], with (x^i, y^i) as the mean vector, and the covariance matrix describing the probability density function's shape.

The resulting distribution is an ellipse centered at (x^i, y^i) with the major and minor axes determined by the covariance matrix. However, we do not blindly accept all of the sampling results; instead, we apply multiple rules to determine their acceptability, inspired by rejection sampling [160]. These rules include verifying that the sampled geometric positions adhere to symbolic relationships at a high level, avoiding object overlap, and ensuring that objects remain within the table boundary. For example, if the bounding box of an object position falls outside the detected table bounds, we reject that sample. The bounding box of objects and the table are computed based on their respective properties, such as size or shape. After multiple rounds of sampling, we can obtain M object configuration sequences.

6.3.3 Computing Task-Motion Plans

After identifying feasible object configurations on the tabletop in Steps 1 and 2, the next step is to place the objects on the tabletop based on one of object configuration sequences. At the task level, the robot must decide the sequence of object placement and how to approach the table. For example, if a bread is on top of a plate, the robot must first place the plate and then the bread. The robot must also determine how to approach the table, such as from which side of the table. Once the task plan is determined, the robot must compute 2D navigation goals (denoted as loc) at the motion level that connect the task and motion levels. Subsequently, the robot plans motion trajectories for navigation and manipulation behaviors.

In the presence of dynamic obstacles, not all navigation goals (loc) are equally preferred. For instance, it might be preferable for the robot to position itself close to an object for placement rather than standing at a distance and extending its reach. A recent approach called GROP [96] was developed for computing the optimal navigation goal loc, which enabled the task-motion plan with the maximal utility for placing each object in terms of feasibility and efficiency given an object configuration (x_j^i, y_j^i), where $0 \leq j \leq M$. Therefore, for different groups of object configurations, we use GROP to compute the maximal utility value of task-motion plans and select the best one for execution. Figure 6.3 shows one task-motion plan generated using LLM-GROP for a four-object rearrangement task.

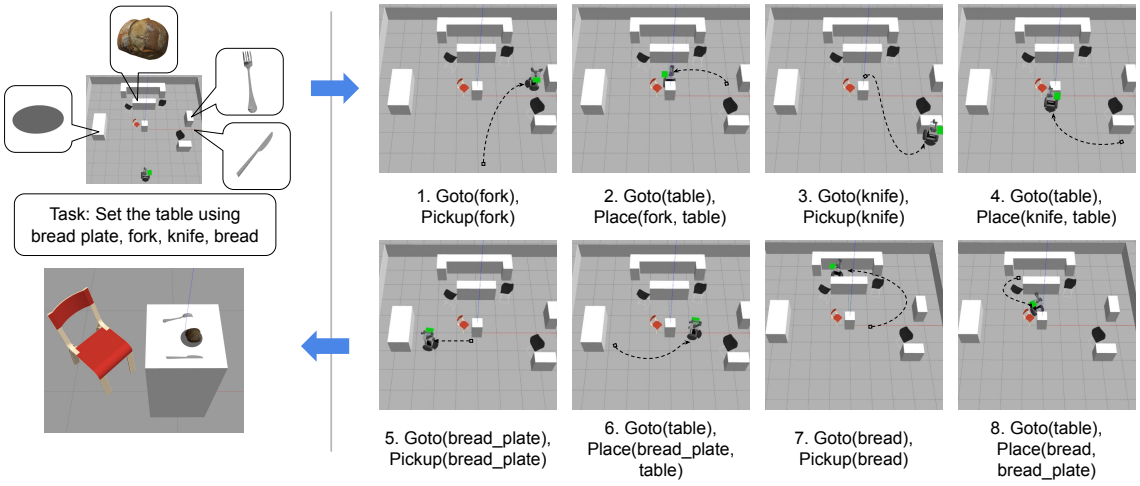


Figure 6.3: An illustrative example of LLM-GROP showing the robot navigation trajectories (dashed lines) as applied to the task of "set the table with a bread plate, a fork, a knife, and a bread." LLM-GROP is able to adapt to complex environments, using commonsense extracted from GPT-3 to generate efficient (i.e., minimize the overall navigation cost) and feasible (i.e., select an available side of the table to unload) pick-and-place motion plans for the robot.

6.4 Experiments

In this section, we evaluate the performance of LLM-GROP using the task of rearranging tableware objects. The robot needs to compute semantically valid tabletop arrangements, plan to efficiently rearrange the objects, and realize the plan via navigation and manipulation behaviors.

6.4.1 Experimental Setup

Baselines: LLM-GROP is evaluated by comparing its performance to three baselines, where the first baseline is the weakest.

- Task Planning with Random Arrangement (TPRA): This baseline uses a task planner to sequence navigation and manipulation behaviors, while it randomly selects stand-

Table 6.1: Objects that are involved in our object rearrangement tasks for evaluation, where tasks 1-5 include three objects, tasks 6 and 7 include four objects, and task 8 includes five objects.

Task #ID	Objects
1	Dinner Plate, Dinner Fork, Dinner Knife
2	Bread Plate, Water Cup, Bread
3	Mug, Bread Plate, Mug Mat
4	Fruit Bowl, Mug, Strawberry
5	Mug, Dinner plate, Mug Lid
6	Dinner Plate, Dinner Fork, Mug, Mug Lid
7	Dinner Plate, Dinner Fork, Dinner Knife, Strawberry
8	Dinner Plate, Dinner Fork, Dinner Knife, Mug, Mug Lid

Table 6.2: Hypermeters of OpenAI's GPT-3 engines in Our Experiment

Parameter	Value	Parameter	Value
Model	text-davinci-003	Temperature	0.1
Top p	1.0	Maximum length	512
Frequency penalty	0.0	Presence penalty	0.0

ing positions next to the target table and randomly places objects in no-collision positions on the table.

- LLM-based Arrangement and Task Planning (LATP): It can predict object arrangements using LLMs and perform task planning. It uniformly samples standing positions around the table for manipulating objects.

- GROP [96]: It considers plan efficiency and feasibility for task-motion planning, and lacks the capability of computing semantically valid arrangements. Similar to TPRA, GROP also randomly places objects in no-collision positions on the table.

Table 6.3: Rating guidelines for human raters in the experiments. **1** point indicates the poorest tableware object arrangement as it suggests that some objects are missing. Conversely, **5** points represent the best arrangement.

Points	Rating Guidelines
1	Missing critical items compared with the objects listed at the top of the interface (e.g., dinner plate, dinner fork, dinner knife), making it hardly possible to complete a meal.
2	All items are present, but the arrangement is poor and major adjustments are needed to improve the quality to a satisfactory level.
3	All items are present and arranged fairly well, but still there is significant room to improve its quality.
4	All items are present and arranged neatly, though an experienced human waiter might want to make minor adjustments to improve.
5	All items are present and arranged very neatly, meeting the aesthetic standards of an experienced human waiter.

A mobile manipulator is assigned the task of setting a dinner table using a specific set of objects. In a simulated environment[1], the robot needs to retrieve multiple objects from various locations and place them on the central table. Additionally, an obstacle (i.e., a chair) will be randomly placed around the table. There are eight tasks that involve handling different objects, as detailed in Table 6.1. We execute each task 20 times using the LLM-GROP system with the same prompt templates, and after each task is completed, we capture an image of the table, the chair, and the objects on the tabletop for later human evaluation. To carry out our experiments, we used OpenAI's GPT-3 engines. Please refer to Table 6.2 for the specific hyperparameters we adopted. We have chosen not to use ChatGPT, a well-known language LLM, for large-scale experiments due to the unavailability of its APIs.

Rating Criteria: We recruited five graduate students with engineering backgrounds, three

[1] Implemented in the Gazebo simulator

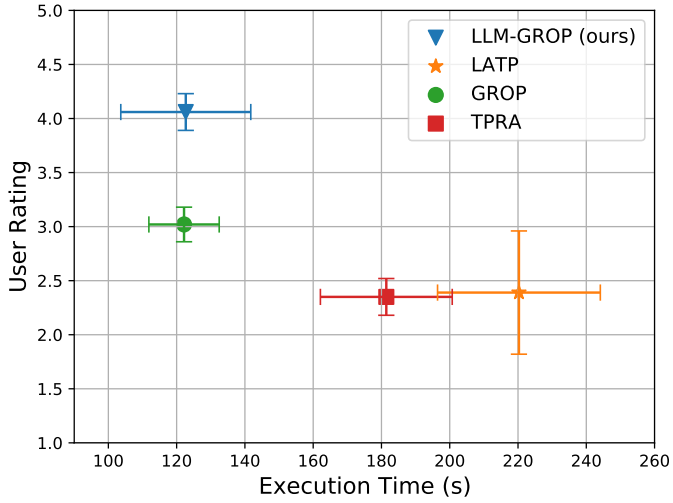


Figure 6.4: Overall performance of LLM-GROP as compared to three baselines based on mean values and standard errors of user ratings and robot execution time for all tableware object arrangement tasks.

females and two males between the ages of 22 and 30. We designed a five-point rating rule, which is outlined in Table 6.3, and tasked the volunteers with scoring tableware object re-arrangements in images they were shown. We generated 640 images from the four methods (three baselines and LMM-GROP) for eight tasks and each image required evaluation from all volunteers, resulting in a total sample size of 3200 images. The volunteers were shown one image at a time on a website[2] that we provided, and they scored each image from 1 to 5 based on the rating rules. We ensured that the rating was rigorous by using a website to collect rating results, thereby minimizing any potential biases that could arise from further interaction with the volunteers once they entered the website.

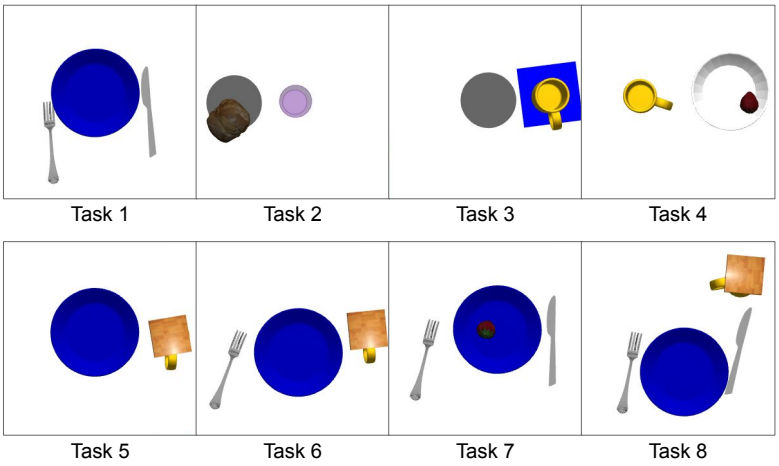


Figure 6.5: Examples of tableware objects rearranged by our LLM-GROP agent in eight tasks, where the objects used in these tasks can be found in Table 6.1. Our LLM-GROP enables the arrangement of tableware objects to be both semantically valid.

6.4.2 Experimental Results

LLM-GROP vs. Baselines: Figure 6.4 shows the key findings of our experiments, which compares the performance of LLM-GROP to the three other baseline approaches. The x-axis indicates the time each method takes to complete a single task, while the y-axis indicates the corresponding user rating. The results demonstrate that our LLM-GROP achieves the highest user rating and the shortest execution time compared to the other approaches. While GROP proves to be as efficient as our approach, it receives a significantly lower rating score. By contrast, TPRA and LATP both receive lower user ratings than our LLM-GROP. They also display poor efficiency. This is because they lack the navigation capabilities to efficiently navigate through complex environments. For instance, when their navigation goals are located within an obstacle area, they struggle to adjust their trajectory, leading to longer task completion times.

[2]The link for the questionnaire-based experiment results evaluation is `http://150.158.148.22/`

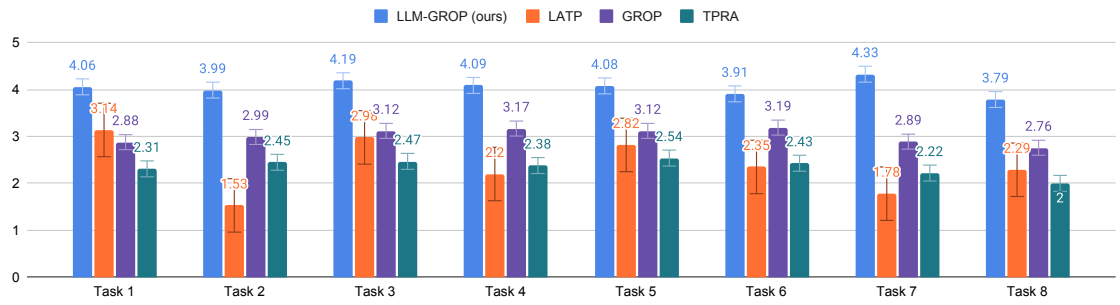


Figure 6.6: User ratings of individual object rearrangement tasks, with the x-axis representing the task and the y-axis representing the user rating score. It can be observed that LLM-GROP consistently performs the best compared to baselines. Tasks 1-5 involve three objects, tasks 6 and 7 involve four objects, and task 8 involves five objects. The numerical value displayed on each bar indicates the mean rating for the corresponding task.

Figure 6.5 provides several examples of various tasks that are rearranged by our agent. Figure 6.6 presents the individual comparison results of each method for individual tasks. The x-axis corresponds to Task #ID in Table 6.1, while the y-axis represents the average user rating for each method. Our LLM-GROP demonstrates superior performance over the baselines for each task. Specifically, tasks 1 to 5 receive slightly higher scores than tasks 6 and 8. This is reasonable because the latter two tasks require the robot to manipulate more objects, posing additional challenges for the robot.

Real Robot Demonstration: We tested our LLM-GROP approach on a real mobile robot platform to demonstrate its effectiveness in rearranging a set of tableware objects, as shown in Figure 6.7. The set included a dinner plate, a dinner fork, a dinner knife, a water cup, and a strawberry. The robot started on the left table and is tasked with rearranging the objects on the right table in the left image. After successfully completing the task, the robot successfully rearranged the objects as shown in the right image. The final object placements were semantically valid, such as the fork being on the left of the dinner plate and

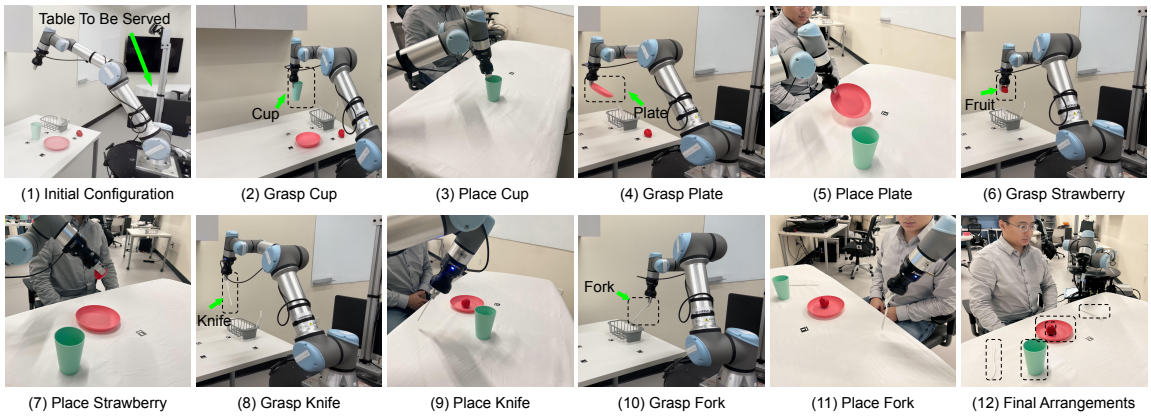


Figure 6.7: We demonstrate LLM-GROP on real robot hardware. The real-robot system includes a Segway-based mobile platform and a UR5e robot arm. The robot employs hard-coded procedures for object grasping. The task is to serve a human with a knife, a fork, a cup, a plate, and a strawberry. The robot computes a plan that successfully avoids chairs and the human around the table, while being able to place the target objects in plausible physical positions.

the strawberry being on the plate. These outcomes effectively demonstrate the effectiveness of our approach in performing real-world tasks using a robotic platform. We have generated a demo video that has been uploaded as part of the supplementary materials.

7 Planning with Large Language Models to Handle Unforeseen Situations

7.1 Introduction

Robots that operate in the real world frequently encounter long-horizon tasks that require multiple actions. Automated task planning algorithms have been developed to help robots sequence actions to accomplish those tasks [75]. Closed world assumption (CWA) is a presumption that was developed by the knowledge representation community and states that "statements that are true are also known to be true" [6]. Most current task planners have been developed for closed worlds, assuming complete domain knowledge is provided and one can enumerate all possible world states [161, 162, 65, 163]. However, the real world is "open" by nature, and unforeseen situations are common in practice [3]. As a consequence, current automated task planners that are largely knowledge-based tend to be fragile in open worlds rife with unforeseen situations. Figure 7.1 shows an example situation: *Aiming to grasp a cup for drinking water, a robot found that the cup was not empty*. Although one can name many such situations, it is impossible to provide a complete list of them. To this end, researchers have developed open-world planning methods for robust task completions

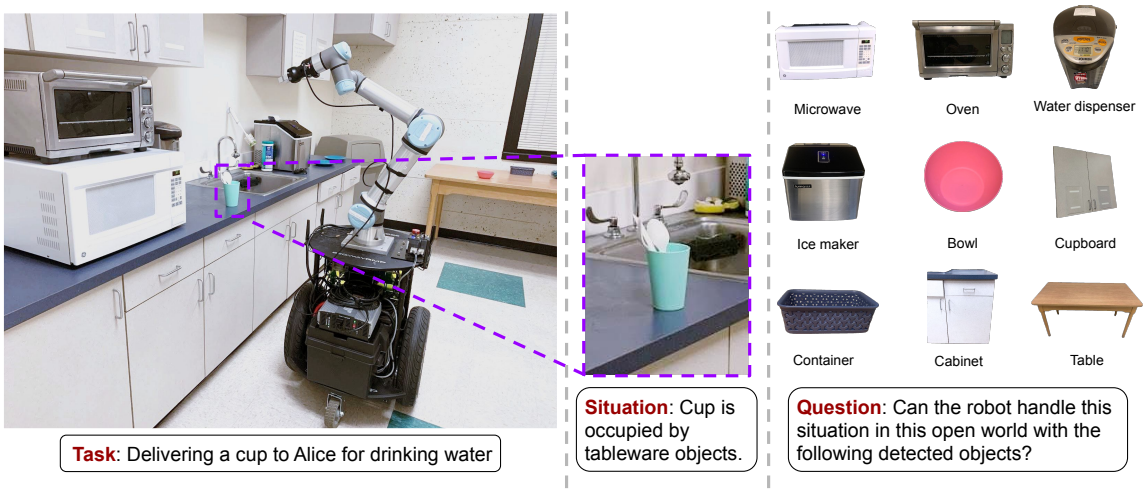


Figure 7.1: An illustrative example of a *situation* in the real world, encountered during the execution of the plan "delivering a cup to a human for drinking water." The robot approached a cabinet in a kitchen room on which a cup was located. The robot then found the cup to be delivered. Before grasping it, however, the robot detected a situation that *the cup was occupied with a fork, a knife, and a spoon*. This situation prevented the robot from performing the current action (i.e., grasping) and rendered normal solutions for drinking water invalid. On the left is a kitchen environment and our mobile manipulator detecting a cup. On the right, we show a few visually perceived objects that were potentially useful for situation handling.

in real-world scenarios [7, 164, 3, 152, 28, 29].

In the current literature, there are at least three ways of addressing the open-world planning problem. The first involves acquiring knowledge via human-robot interaction, e.g., dialog-based, to handle situations in an open-world context [165, 69, 166]. Those methods require human involvement, which might hinder their autonomy capability and limit their applicability in the real world. A second idea for open-world planning relies on dynamically building a knowledge base to assist a pre-defined task planner, where this knowledge base is usually constructed in an automatic way using external information, e.g., using open-source knowledge graphs [7, 164, 3]. Such external knowledge bases are considered "bounded" in this work due to their representation and knowledge source, which limits the

"openness" of their task planners. Large Language Models (LLMs) have been developed in recent years [140, 151, 14, 167], and those LLMs have demonstrated major improvements in a variety of downstream tasks. Researchers have investigated the idea of extracting common sense from LLMs to guide robot task planning [29, 28, 168, 152]. One challenge in this process is that the knowledge from LLMs is *domain-independent* [169, 170], whereas the robot faces specific domains that are featured with many *domain-dependent* constraints. For example, an LLM may provide a robot with the knowledge of how to serve water to people, but it falls short in determining the available types of containers (such as cups or glasses) the robot has access to, as well as the water sources in stock (like tap water or bottled water). In line with the third idea, we use LLMs for open-world planning in this work. To address the challenge of grounding common sense to specific domains, we propose to enable the marriage of classical AI planning methods, and LLM-based knowledge extraction.

In this work, we develop a robot task planning and situation handling framework, called *Common sense-based Open-World Planning* (**COWP**), that uses an LLM for dynamically augmenting automated task planners with external task-oriented common sense. COWP is based on classical planning and leverages LLMs to augment action knowledge (action preconditions and effects) for task planning and situation handling. The **main contribution** of this work is a novel integration of a pre-trained LLM with a knowledge-based task planner. Inheriting the desirable features from both sides, COWP is well grounded in specific domains while embracing commonsense solutions at large.

To conduct systematic evaluations, we selected 12 dining-focused tasks from the *ActivityPrograms* dataset [15]. Each task consisted of a high-level name, such as "Serve water",

and a natural language description of the action sequence required to complete it. Using a crowdsourcing platform, we recruited 112 participants to propose potential situations that might hinder successful task execution. We gathered a situation dataset that includes more than one thousand situations for evaluation purposes. According to experimental results, we see COWP performed better than literature-selected baselines [7, 28, 31] in terms of the respective success rates in task completion and situation handling. We implemented and demonstrated COWP using a mobile manipulator.

7.2 Related Work

In this section, we first briefly discuss classical task planning methods that are mostly developed under the closed world assumption. We then summarize three families of open-world task planning methods for robots, which are grouped based on how unforeseen situations are addressed.

Classical Task Planning for Closed Worlds: A classical task planning problem consists of two main components: a domain description and a problem description [76]. The domain description includes a collection of actions, each of which is defined by its preconditions and subsequent effects. The problem description, on the other hand, specifies the initial state and the desired goal conditions. A sequence of actions can be generated, enabling an effective transition from the initial state to the designated goal state.

Closed world assumption (CWA) indicates that an agent is provided with complete domain knowledge, and that all statements that are true are known to be true by the agent [6]. In this work, such a classical planning system is referred to as a closed-world task plan-

ner. Although robots face the real world that is open by nature, their planning systems are frequently constructed under the CWA [3, 70, 171, 75, 76, 65]. The consequence is that those robot planning systems are not robust to unforeseen situations at execution time. In this work, we aim to develop a task planner that is aware of and able to handle unforeseen situations in open-world scenarios.

Open-World Task Planning with Human in the Loop: Task planning systems have been developed to acquire knowledge via human-robot interaction to handle open-world situations [165, 69, 166]. For instance, researchers created a planning system that uses dialog systems to augment their knowledge bases [165], whereas Amiri et al. (2019) further modeled the noise in language understanding [69]. Tucker et al. (2020) enabled a mobile robot to ground new concepts using visual-linguistic observations, e.g., to ground the new word "box" given command of "move to the box" by exploring the environment and hypothesizing potential new objects from natural language [166]. The major difference from those open-world planning methods is that COWP does not require human involvement.

Open-World Task Planning with External Knowledge: Some existing planning systems address unforeseen situations by dynamically constructing an external knowledge base for open-world reasoning. For instance, researchers have developed object-centric planning algorithms that maintain a database about objects and introduce new object concepts and their properties (e.g., location) into their task planners [7, 164]. For example, Jiang et al. (2019) developed an object-centric, open-world planning system that dynamically introduces new object concepts through augmenting a local knowledge base with external information [7]. In the work of Hanheide et al. (2017), additional action effects and assumptive actions were

modeled as an external knowledge to explain the failure of task completion and compute plans in open worlds [3]. A major difference from their methods is that COWP employs an LLM as a generative approach that is capable of responding to any situation, whereas the external knowledge sources of those methods limits the openness of their systems.

Closed-World Task Planning with LLMs: A straightforward idea for LLM-based planning is to directly enter planning domain information into LLMs, and request plan generations. Recent research has shown that a naive implementation of such ideas produces very poor performance even if the planning domain is as simple as Blocksworld [172, 173]. The ChatGPT report (in its conclusion section) also identified "long-horizon planning" as a challenging task, and encouraged more research on LLM-based planning [174]. Very recently, researchers have investigated translating descriptions of planning tasks in natural language into PDDL [76], and then computing plans using PDDL systems [34]. The produced system called LLM+P takes natural language descriptions as input, and computes plans with optimality guarantee. The action knowledge of LLM+P formulated in PDDL was provided by domain experts, and could not adapt to novel situations at execution time. By comparison, COWP (ours) dynamically extract common sense from LLMs for augmenting its action knowledge for planning and situation handling.

Open-World Task Planning with LLMs: In recent years, many LLMs have emerged, including BERT [25], GPT-3 [140], ChatGPT [14], CodeX [150], and OPT [151]. These LLMs encode a large amount of commonsense knowledge [27, 175, 176, 177] and have been employed in robot task planning [152, 28, 29, 30, 31, 10]. For instance, the work of Huang et. al. [28] showed that LLMs can effectively facilitate task planning in household

environments by iteratively augmenting prompts [28]. The SayCan system enabled robot planning with affordance functions to account for action feasibility, where the service requests are specified in natural language (e.g., "deliver a Coke") [29]. Additionally, various teams also have successfully applied LLMs to generate plans for high-level tasks expressed in natural language by sequencing actions [30, 152, 8, 178, 179, 10].

It is generally difficult to ground LLM-based planning systems in the real world as the LLMs were not trained for specific domains. For instance, LLM-based planners frequently generate plans that require objects or tools that are not present in the scene [30]. COWP (ours) alleviates this issue through reasoning about robot skills using rule-based action knowledge.

Work closest to COWP is a recent LLM-based planning system, called ProgPrompt [31], which generates task plans and handles situations using programmatic LLM prompts. ProgPrompt handles situations by asserting preconditions of the plan (e.g., being close to the fridge before attempting to open it) and responding to failed assertions with appropriate recovery actions. Compared with ProgPrompt, which relies on example solutions in prompting to guide the LLMs, COWP (ours) uses action knowledge to enable zero-shot prompting for planning and situation handling. For example, in the case of ProgPrompt, three examples are typically provided in the prompt. These examples guide ProgPromptto generate task plans, and these plans take into account feedback from the environment by including preconditions for actions. By comparison, COWP makes use of action knowledge, which can be in the form of rules, facts, and principles related to the task. This can include a detailed understanding of how actions impact the state of the environment, which is not required in the prompts used by ProgPrompt. This enables COWP to create task plans

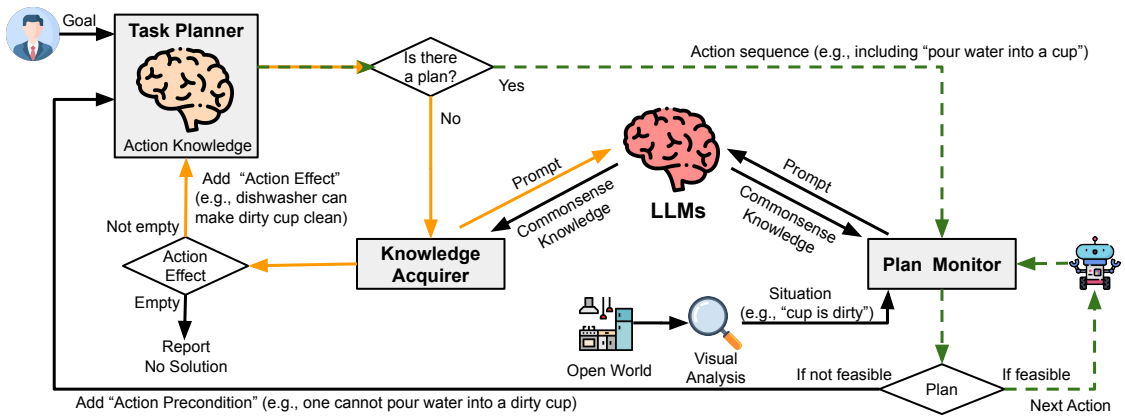


Figure 7.2: An overview of **COWP** that includes the three key components of Task Planner (provided as prior knowledge under closed-world assumption), Knowledge Acquirer, and Plan Monitor. The **green** (dashed) loop represents a plan execution process where the robot does not encounter any situation, or these situations have no impact on the robot's plan execution. The **orange** loop is activated when the robot's current (closed-world) task planner is unable to develop a plan, which activates Knowledge Acquirer to augment the task planner with additional action effects utilizing common sense.

without the need for example solutions, i.e., zero-shot prompting.

COWP extracts commonsense knowledge from LLMs and incorporates rule-based action knowledge from human experts. Reasoning with action knowledge ensures the soundness of task plans generated by COWP, while querying LLMs guarantees the openness of COWP to unforeseen situations. As a result, COWP can be better grounded to specific domains, and is able to incorporate common sense to augment robot capabilities supported by predefined skills.

7.3 Algorithm

In this section, we first provide a problem statement and then present our open-world planning approach called Common sense-based Open-World Planning (COWP).

Our goal is to address open-world planning problems for robots. An open-world plan-

Algorithm 5 COWP algorithm

Require: Task Planner, Plan Monitor, Knowledge Acquirer, and an LLM
Input: A domain description, and a problem description

```
1: while task is not completed do
2:     Compute a plan pln using Task Planner given the domain description and the problem description
3:     for each action in pln do
4:         Evaluate the feasibility of the current plan using Plan Monitor given a situation
5:         if a plan is not feasible then
6:             Add an action precondition to Task Planner
7:             Compute a new task plan pln', and pln ← pln'
8:             if pln is empty then
9:                 Extract common sense (action effects) using Knowledge Acquirer
10:                if action effect is not empty then
11:                    Add an action effect to Task Planner
12:                    Compute a new task plan pln'', and pln ← pln''
13:                else
14:                    Report "no solution"
15:                end if
16:            end if
17:        else
18:            Execute the next action by the robot
19:        end if
20:    end for
21: end while
```

ning problem assumes that the robot might encounter situations that are not considered in the development of the planning systems. More specifically, we use the term of *situation* to refer to an unforeseen world state that potentially prevents an agent from completing a task using a solution that normally works. In this work, we assumed the availability of a description of the robot's skills, formulated in action description language PDDL. PDDL is designed to formalize AI planning problems, allowing for a more direct comparison of planning algorithms and implementations [18]. The *objective* of an open-world planner is to compute plans that can adapt to and handle unexpected situations while pursuing the completion of service tasks or reporting "no solution" when appropriate.

7.3.1 Algorithm Description

Figure 7.2 illustrates the three major components of our COWP framework. **Task Planner** is used for computing a plan under the closed-world assumption and is provided as prior knowledge in this work. **Plan Monitor** evaluates the overall feasibility of the current plan using common sense. **Knowledge Acquirer** is for acquiring common sense to augment the robot's action effects when the task planner generates no plan.

Algorithm 5 describes how the components of COWP interact with each other. Initially, Task Planner generates a satisfying plan based on the goal provided by a human user in **Line 2**. After that, the actions in the plan are performed sequentially by a robot in the for-loop of **Lines 3-20**. If the current plan remains feasible, as evaluated by Plan Monitor, the next action will be directly passed to the robot for execution in **Line 18**; otherwise, an action precondition will be added to Task Planner in **Line 6**. For example, when Task Planner does not know anything about "dirty cups," it might wrongly believe that one can use a dirty cup as a container for drinking water. In this situation, **Line 6** will add a new statement "*The precondition of filling a cup with water is that the cup is not dirty*" into the task planner. After the domain description of Task Planner is updated (adding action preconditions), the planner tries to generate a new plan pln' in **Line 7**. If no plan is generated by Task Planner (**Line 8**), Knowledge Acquirer will be activated for knowledge augmentation with external task-oriented common sense in **Line 9**. If the extracted common sense includes action effects, such information will be added into the task planner in **Line 11**. For instance, the robot might learn that a chopping board can be used for holding steak, as an action effect that was unknown before. This process continues until no additional action effects can be

generated (in this case, COWP reports "no solution" in **Line 14** or a plan is generated.

COWP leverages common sense to augment a knowledge-based task planner to address situations towards robust task completion in open worlds. The implementations of **Lines 4** and **9** using common sense are non-trivial in practice. Next, we describe how commonsense knowledge is extracted from an LLM for those purposes.

7.3.2 Plan Monitor and Knowledge Acquirer

In this section, two important components of COWP are discussed, i.e., Plan Monitor and Knowledge Acquirer.

Plan Monitor is designed to evaluate if there are any situations that could prevent a robot from completing its current task successfully. To achieve this, the plan monitor takes in action sequences generated from a classical planner and a set of situations collected from the real world. The Plan Monitor uses a prompt template to query an LLM for each action in the sequence. The prompt template is based on the following structure:

Prompt 1: *Is it suitable for a robot to [Perform-Action], if [Situation]?*

The placeholder [] represents the specific information to be filled in within the template. In this case, *[Perform-Action]* represents a natural language description of an action generated from our PDDL-based task planner, with the action in the form of "Action Object-1 Object-2 ...". *[Situation]* represents a natural language description of a situation. For example, if the current action is "Fill Cup Water" and the given situation is "Cup is broken", the corresponding prompt would be "*Is it suitable for a robot to fill a cup with water, if the cup is broken?*". Translating a symbolic action into a natural language description can be

achieved either by following handcrafted rules or with the assistance of LLM's grammar completion. In this case, we adopt the first approach. An LLM generates a natural language response to each prompt. If the LLM determines that the action is infeasible given the situation described, the whole plan is considered infeasible.

Knowledge Acquirer aims to acquire common sense for augmenting the task planner's action knowledge for situation handling. In particular, the input for the knowledge acquirer consists of a collection of available objects in the environment that the robot can access. Another template-based prompt is developed for querying an LLM for acquiring common sense about action effects.

Prompt 2: *Is it suitable for a robot to [Perform-Action-with-Object]?*[1]

In the prompt, the placeholder *[Perform-Action-with-Object]* represents a natural language description of a robot performing an action with another object. For example, if the action "Fill Cup Water" is considered infeasible under the situation "Cup is broken" by Plan Monitor, one instance of using Prompt 2 with another object "bowl" is "*Is it suitable for a robot to fill a bowl with water?*". If the response from an LLM to the example prompt is "*Yes*", an additional action effect, "The bowl can also be a container to fill water", will be added to the task planner.

Continuing the "Serve water" example, additional action effects introduced through Prompt 2 might enable the task planner to generate many feasible plans, e.g., using a glass, measuring cup, or bowl to fill water. It can be difficult for the task planner to evaluate

[1]Intuitively, the solution from COWP goes beyond finding alternative objects in addressing unforeseen situations. COWP enables situation handling by manipulating the attributes of individual instances. For example, a "dirty cup" situation can be handled by running a dishwasher, where no second object is involved.

which plan makes the best sense. To this end, we develop Prompt 3 for selecting a task plan of the highest quality among those satisficing plans:

> Prompt 3: *There are some objects, such as [Object-1], [Object-2], ..., and [Object-N]. Which is the most suitable for [Current-Task], if [Situation]?*

The placeholder *[Current-Task]* is a high-level task description in natural language. In the example of "Serve water", we expect the LLM to respond by suggesting "glass" being the most suitable for holding water among those mentioned items.

7.3.3 Implementation

Here, we explain how to implement COWP using the "Serve water" task as an example. To do so, we use a PDDL-based closed-world task planner, which requires both a domain file and a problem file. The domain file defines a set of predicates (e.g., `is_grasped`) and actions (e.g., *fill*), with each action specified by its preconditions and effects. Consider the following action definition for *fill* shown in Figure 7.3, which includes preconditions like `(is_grasped ?c)` ∧ `(is_empty ?c)` and effects such as `(is_filled ?c)`.

The problem file, on the other hand, defines the task by specifying an initial state and a goal state, which in this case is "Water is served to the user". To generate a task plan, a solver, such as Fast Downward [163] can be used. A feasible closed-world plan for this task is shown in Figure 7.4.

Plan Monitor is responsible for checking action feasibility under a given situation, such as "Cup is dirty". If an action, like *fill*, is considered infeasible by an LLM, a constraint, such as `not (is_dirty ?c)`, will be added to the action precondition. We implement

```
(:action fill
        :parameters
        (?r - robot ?c - cup
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?c)
             (is_empty ?c)
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?c)
             (not (is_on ?f))
             (is_off ?f))
)
```

Figure 7.3: Definition of action "*fill*" in our PDDL-based task planner

```
S1: find robot cup kitchen
S2: find_faucet robot faucet kitchen
S3: turnon robot faucet kitchen
S4: grasp robot cup kitchen
S5: fill robot cup faucet kitchen
S6: move robot cup kitchen dining
S7: place robot cup table dining
```

Figure 7.4: One closed-world plan for the task "Serve water"

a predicate generator that translates the natural language situation into a symbolic form, such as `is_dirty ?cup`. This generator utilizes the few-shot learning capabilities of the LLM. Once the predicate is obtained, it is added to the PDDL code and highlighted with an underline, as shown in Figure 7.5. Additionally, the initial state in the problem file is updated to reflect the new situation, `(is_dirty cup)`.

After adding the constraint `not (is_dirty ?c)`, the planning system might not generate a feasible plan, as the object *cup* is found to be dirty and cannot be used. In such cases, the planning system needs to seek alternative solutions to accomplish the task by

```
(:action fill
        :parameters
        (?r - robot ?c - cup
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?c)
             (is_empty ?c)
             (not (is_dirty ?c))
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?c)
             (not (is_on ?f))
             (is_off ?f))
)
```

Figure 7.5: An action constraint, `not (is_dirty ?c)`, is added into the action "*fill*"

adding action effects to the planner. The commonsense knowledge that "Bowl can be a container to fill water" is obtained by Knowledge Acquirer. This knowledge might have been overlooked by the planning system developer. COWP can add this new action effect into the planning system, as shown in Figure 7.6. Now, the *fill* action can be applied to both *cup* and *bowl*. Additionally, related parts of the code are adjusted accordingly, and the initial state in the problem file is updated to include the bowl's location.

An alternative plan that uses a bowl for serving water, shown in Figure 7.7, can complete the service task "Serve water" and handle the situation "Cup is dirty".

7.4 Experiments

In this section, we evaluate COWP's performance in planning and situation handling.

```
(:action fill
        :parameters
        (?r - robot ?b - bowl
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?b)
             (is_empty ?b)
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?b)
             (not (is_on ?f))
             (is_off ?f))
)
```

Figure 7.6: The PDDL-based task planner incorporates the commonsense knowledge that "Bowl can be a container to fill water" as an action effect.

```
S1: find robot bowl kitchen
S2: find_faucet robot faucet kitchen
S3: turnon robot faucet kitchen
S4: grasp robot bowl kitchen
S5: fill robot bowl faucet kitchen
S6: move robot bowl kitchen dining
S7: place robot bowl table dining
```

Figure 7.7: A feasible plan for the task "Serve water", which can complete the service task "Serve water" and handle the situation "Cup is dirty."

7.4.1 Experimental Setup

Our experiments were performed in a *dining domain*, where a service robot is tasked with fulfilling a user's service requests in a home setting. For simulating dining domains, we chose 12 everyday tasks from the *ActivityPrograms* dataset [15]. These tasks can be found in Table 7.1. For each task, we constructed PDDL-based planning systems to generate action sequences for their completion.

To carry out our experiments, we used OpenAI's GPT-3 engines. Please refer to Ta-

Table 7.1: 12 tasks extracted from the ActivityPrograms dataset [15] for evaluation purposes.

Task Name	Task Name	Task Name
Set table	Serve water	Serve coke
Wash plate	Heat burger	Make coffee
Clean floor	Prepare burger	Store food
Wash cup	Wash sink	Wash glass

Table 7.2: Hypermeters of OpenAI's GPT-3 engines in Our Experiment

Parameter	Value
Model	text-davinci-003
Temperature	0.0
Top p	1.0
Maximum length	32
Frequency penalty	0.0
Presence penalty	0.0

ble 7.2 for the specific hyperparameters we adopted. In simulation experiments, we assume that our robot is provided with a perception module for converting raw sensory data into logical facts (situations), such as objects and their properties, while the robot still needs to reason about the facts for planning and situation handling. Our experiments were performed using simulated robot behaviors and situations collected from human participants.

7.4.2 Simulation Platform

Situation Dataset: To evaluate the performance of COWP in dealing with situations in a dining domain, we collected a dataset of execution-time situations using Amazon Mechanical Turk. Each instance of the dataset corresponds to a situation that prevents a service robot from completing a task. We recruited MTurkers with a minimum HIT score of 70.

Each MTurker was provided with a task description, including steps for completing the task. The MTurkers were asked to respond to a questionnaire by identifying one step in the provided plan and describing a situation that might occur in that step. For example, in the task "Serve water", we provided its plan consisting of the following steps: "1) Walk to kitchen room, 2) Find glass, 3) Find sink, 4) Find faucet, 5) Turn on faucet, 6) Fill glass with water, 7) Move glass near table, 8) Place cup on table". Two common responses from the MTurkers were "*Glass is broken*" for Step 2 and "*Faucet has no water*" for Step 5.

We received 1,224 responses from MTurkers, and 1,085 of them were valid, where the responses were evaluated through both a validation question and a manual filtering process. We included a validation question that resembled other questions in the middle of the questionnaire, but instead of asking for a situation, it required the MTurker to input a provided "secret code", such as "Successfully Verified!". Regarding the manual filtering process, we checked the responses for relevance to the task and logical coherence. For instance, some MTurkers copied and pasted text that was completely irrelevant, and those responses were removed manually. Additionally, while some responses like "Glass's color is green" were related to the task, such as "Serve water," they were considered invalid because the robot can still complete the task regardless of the glass's color. There are at least 88 situations collected for each of the 12 tasks. To facilitate the construction of the simulation platform, we further grouped the situations of significant similarities and generate a set of "distinguishable" situations. For instance, two situations, "Glass is broken" and "Glass is shattered" are considered indistinguishable. As a result, each task has 12 to 24 distinguishable situations, and the specific number of distinguishable situations for each task can be found in Table 7.3. Here, we show 15 distinguishable situations for the task "Serve water":

1) *Glass is broken.*

2) *Faucet has no water.*

3) *Glass is dusty.*

4) *Glass is missing.*

5) *Water is dirty.*

6) *Faucet has sustained physical damage.*

7) *Faucet cannot be turned on.*

8) *Sink is not found.*

9) *Faucet is leaking.*

10) *Faucet is not found.*

11) *Water spills on floor.*

12) *Glass is not full of water.*

13) *Kitchen door is locked and cannot be opened.*

14) *Glass falls onto floor.*

15) *Glass is stuck and cannot be removed.*

On our project website (`https://cowplanning.github.io/`), users can download both the MTurk questionnaire and the situation dataset. The dataset is provided as a CSV file comprising 12 separate sheets, each representing the situations for a distinct task. The name of the task corresponds to the sheet name. Each sheet consists of five columns. The situations' descriptions provided by the MTurkers are in Column A. Column B details the corresponding steps where the described situation occurs. Column C is the index of distinguishable situations, while Column D provides descriptions of these situations. Finally, Column E indicates the number of distinguishable situations.

Note that **this dataset is intended solely for evaluation purposes**. The LLMs utilized in COWP act as zero-shot learners, and Plan Monitor can successfully perform its intended function without situation examples.

Table 7.3: The number of distinguishable situations for each task.

Task Name	Num	Task Name	Num
Make coffee	24	Set table	16
prepare burger	21	Clean floor	15
Heat burger	19	Serve water	15
Wash sink	18	Wash cup	15
Store food	17	Serve coke	14
Wash plate	16	Wash glass	12

Simulator: We have constructed a simulator capable of generating a high-level task name (e.g., "Serve water") and an unexpected situation (e.g., "Glass is broken") that occurs during one of the steps (e.g., "Fill glass with water") in each trial. Here, we assume that the probability of the robot encountering a situation while executing an action is denoted by P (i.e., 0.1). This assumption implies that a longer task plan may be more likely to encounter such situations. The simulator also contains 86 objects, including cups, burgers, forks, tables, and chairs. In each trial, it randomly selects and spawns half of the available objects for the robot to manipulate in order to resolve the situation. This setting helps create a diverse environment for the robot. These objects are also extracted from the ActivityPrograms dataset [15] and can be classified into five categories: kitchenware, appliance, furniture, food, and drink. The "kitchenware" category comprises the highest number of objects (29), while the "drink" category contains the fewest (8). For more details, please visit our website at `https://cowplanning.github.io/`.

Baselines and Evaluation Metrics: The evaluation of open-world planners is based on the respective *success rates* of a robot completing service tasks and handling situations in

a dining domain. The following five baselines have been used in our experiments:

- Inner Monologue [30] leverages environmental feedback to generate task plans and handle situations. In the original implementation, this approach could process three types of textual feedback: Passive Scene Description, Success Detection, and Active Scene Description. Passive Scene Description involves obtaining feedback, such as details regarding available objects and situational context, from the environment. One instance of this feedback could be "cup, the cup is broken, and drinking glass." Success Detection checks if every action in the generated plan is successfully executed. However, we have deactivated Active Scene Description because our system excludes humans from the loop. The specific prompts used in Inner Monologue are available in the supplementary material on our project website.

- Closed World (CW) corresponds to classical task planning developed for closed-world scenarios. In practice, CW was implemented by repeatedly activating the closed-world task planner and updating the current world state after executing each action. CW does not have the capability to handle situations. For example, CW is unable to generate a plan that involves using a bowl as a container for serving water.

- External Knowledge (EK) [7] is a baseline approach that enables a closed-world task planner to acquire knowledge from an external source. In our implementation, this external source provides information about a half of the domain objects. For instance, EK may generate a plan that involves using a bowl as a container for serving water, if its knowledge base contains "Bowl can be used for serving water".

- Language Models as Zero-Shot Planners (LMZSP) [28] is a baseline that leverages LLM to compute task plans, where domain-specific action knowledge is not utilized. The LMZSP baseline [28] was not grounded. As a result, their generated plans frequently involve objects unavailable or inapplicable in the current environment. Furthermore, LMZSP cannot receive feedback from its environment, restricting its capability to handle situations. The hyperparameters used in LMZSP are available in the provided supplementary material on our project website.

- ProgPrompt [31] serves as the baseline method that utilizes a programmatic LLM prompt structure to facilitate open-world task planning. The method consists of two types of prompts, namely PROMPT for Planning and PROMPT for State Feedback. ProgPrompt is a few-shot learning method, unlike LMZSP. This is a *competitive* baseline. More specifically, ProgPrompt relies on the "PROMPT for Planning" to create a plan based on the task specification, including a contextual description of the situation. The plan takes feedback from the environment into account by including preconditions for actions. For instance, it might include situated state feedback like "assert ('dirty' to 'cup') else: find('drinkingglass')". If the state is "cup is dirty", the "find drinking glass" action will be executed. The 'State Feedback Prompt' uses feedback from the environment to trigger preconditions in the generated plan. In this example, a context description like "cup is dirty" would trigger the "assert ('dirty' to 'cup')" precondition. This is a competitive baseline. ProgPrompt has the same access to feedback inputs from the environment as our proposed approach, ensuring a fair comparison between the two methods. For more detail on the exact prompts

we used in our research with ProgPrompt, please refer to the provided supplementary materials on our project website.

Baselines ProgPrompt, Inner Monologue and LMZSP have utilized the GPT-3 in their experimental implementations. It is crucial to note that GPT-3 offers multiple models, including "*text-davinci-002*" or "*text-davinci-003*". However, the specific model used by these studies was not mentioned. To ensure a fair comparison, we have chosen to use the "*text-davinci-003*" model for all approaches.

Success Criteria: Unlike many classical planning systems [95, 7, 1, 180] that are guaranteed to provide sound solutions, LLM-based planning systems might generate invalid solutions without being aware of them. For instance, GPT-3 might suggest that one can use a pan for drinking water, which is technically possible, but very uncommon in our everyday life. In this work, we intend to consider those to be unsuccessful trials. LLM-based task planners might wrongly believe that a good-quality plan is generated, while it is actually not the case. To compare with the ground truth, we recruited a second group of people, including six volunteers, to evaluate the performance of different open-world task planners. The volunteers included two females and four males aged 20-40, and all were graduate students in engineering fields. A successful task plan is defined as one that fulfills a user's service request through actions that the robot is capable of executing and are acceptable to humans. In addition, common factors that could render a plan unsuccessful include, but are not limited to, the inclusion of non-existent objects in the environment, actions that the robot is unable to execute, or missing essential steps between actions. Detailed instructions provided for the volunteers prior to the evaluation are available in the provided supplemen-

tary materials on our project website.

Table 7.4: The overall performances of COWP (ours) and two baseline methods are compared in terms of their task completion and situation handling percentages. The task completion percentage is calculated by dividing the number of completed tasks by the total number of trials, which is 150. The situation handling percentage is the ratio of the number of handled situations to the total number of situations.

	COWP (ours)	**ProgPrompt**	**Inner Monologue**
Task completion percentage (%)	**67.8**	61.8	54.0
Situation handling percentage (%)	**36.9**	30.1	22.1

	EK	**CW**	**LMZSP**
Task completion percentage (%)	55.1	44.3	10.2
Situation handling percentage (%)	17.3	0.0	0.0

7.4.3 Experimental Results

COWP vs. Baselines (Overall): Table 7.4 shows the overall performance of COWP in task planning and situation handling, as compared to five baselines, i.e., ProgPrompt, Inner Monologue, EK, LMZSP, and CW. The table demonstrates that COWP outperforms all five baselines in terms of task completion and situation handling percentages. However, LMZSP and CW show poor performance. We believe the poor performance of CW is caused by its inability to leverage external information to handle unforeseen situations. For LMZSP, its poor performance is caused by its weakness in grounding general commonsense knowledge in specific domains and the big noise in the generated plans. As a result, many "solutions" generated by LMZSP are not executable by the robot. For instance, in

one plan, “Find cup; Grab cup; Walk to sink; Run to cup; Walk to water; Find water; Grab water; Pour water into cup” generated by LMZSP, some essential steps were omitted, such as “Walk to kitchen” and “Turn on faucet”. Additionally, the robot was unable to execute the “Grab water” action. This limitation of LMZSP has also been observed and analyzed in the work by Singh et al. [31].

The performance of ProgPrompt is is not as good as that of COWP. This can be attributed to the fact that the task plans produced by ProgPrompt might include noise, such as logical flaws, which reduces the chance of successfully completing the task. For example, one plan generated by ProgPrompt includes the two consecutive actions of “fill glass with water” and “turn on faucet”, which is invalid, because “turn on faucet” should be executed before “fill glass with water.” Additionally, we find the ProgPrompt’s capability relies on the provided examples in the prompt, while COWP (ours) handles situations via zero-shot prompting of LLM.

Inner Monologue’s performance falls short when compared to COWP (ours) and ProgPrompt. Our observation indicates that Inner Monologue tends to generate incomplete task plans and exhibit a bias toward “Bounding Thinking,” which consequently leads to a decrease in both task completion and situation handling percentages. Take, for instance, an Inner Monologue-produced task plan for serving water: “Step 0: walk to kitchen; Step 1: find drinking glass; Step 2: fill drinking glass with water; Step 3: find kitchen table; Step 4: put drinking glass on kitchen table; Step 5: done”. This plan omits the crucial step of grasping the drinking glass first before filling it with water (Step 2), and overlooks the necessary action to access water, like “Switch on Faucet”, given the environment does not have water readily available. This lack of completeness compromises the robot’s ability to

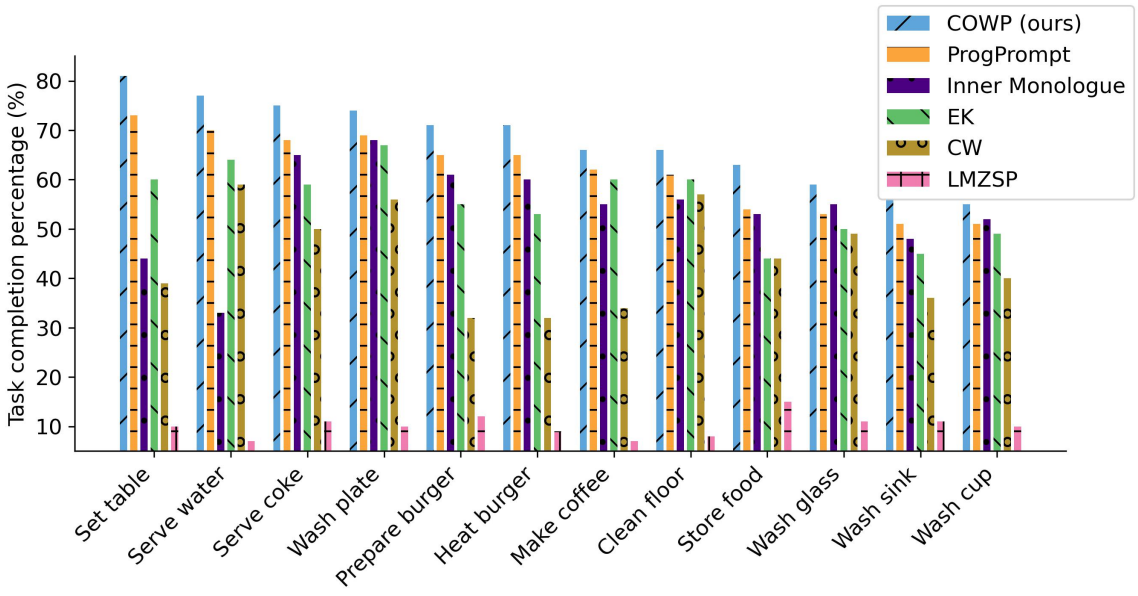


Figure 7.8: The task completion percentage of COWP (ours) and five baseline methods under **12 different tasks**. The *x-axis* represents the task name, and the *y-axis* represents the task completion percentage. The task completion percentage for each value is an average of 150 trials. The tasks are sorted based on the performance of COWP, where the very left corresponds to its best performance.

execute the task. Furthermore, Inner Monologue has been shown to be lacking in situation handling. Despite being informed about situations through Passive Scene Description - a method of gathering environmental context, it often disregards this crucial contextual information. For instance, Inner Monologue may recommend using a cup to serve water even if the "cup is dirty." Moreover, it struggles to identify suitable solutions to address the situation. In the same scenario where a dirty cup is present, Inner Monologue might suggest using a water boiler or a bucket to serve water to humans, even when more suitable options, such as a mug or a clean drinking glass, are available in the environment.

COWP vs. Baselines by Task: Figure 7.8 shows the results of comparing COWP and five baselines in the success rate of task completion under *different tasks*. The *x*-axis denotes

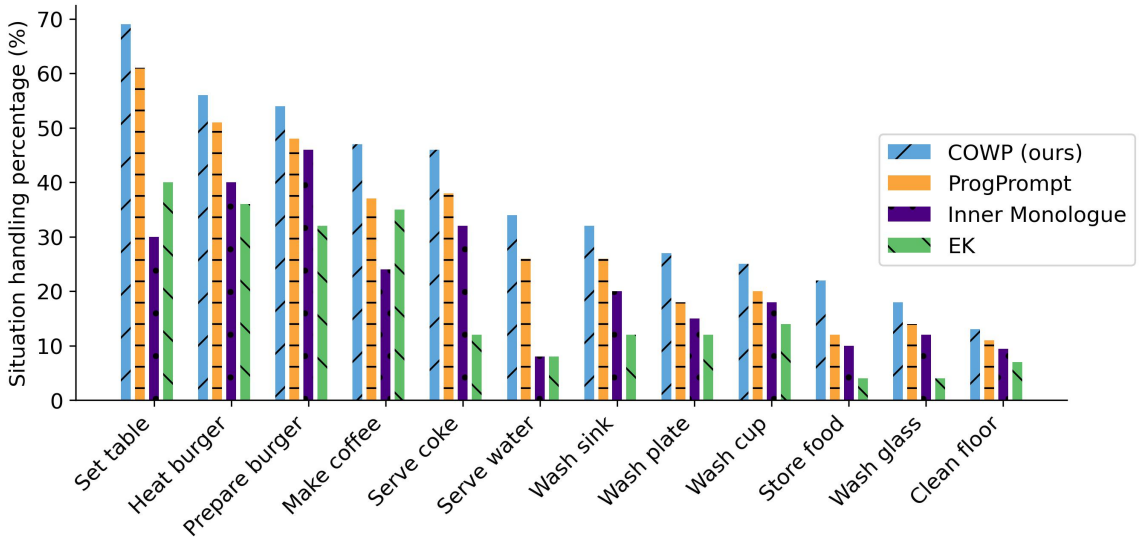


Figure 7.9: The situation handling percentage of COWP (ours) and three baseline methods under **12 different tasks**, where the *x-axis* represents the task name, and the *y-axis* represents the situation handling percentage. Each y value represents a ratio of the number of handled situations to the total number of situations. The tasks are ranked based on the performance of COWP, where the very left corresponds to its best performance.

the task name, and the y-axis indicates the percentage of task completion. From the figure, we can see COWP outperforms the five baselines in all tasks. It is quite interesting to see that open-world planners (including COWP) work better in some tasks than the others. For instance, in the "Set table" task, there are many "missing object" situations, and it happened that the robot could easily find alternative tableware objects to address those situations with the assistance of GPT-3. However, some tasks such as "Wash glass" are more difficult because many situations are beyond the robot's capabilities, e.g., "There is a power outage" and "Faucet has no water".

Figure 7.9 shows the results of comparing COWP and three baselines in the success rate of situation handling under *different tasks*. In this case, we do not include the baseline CW into the figure, because it's inapplicable to the task of situation handling. The results

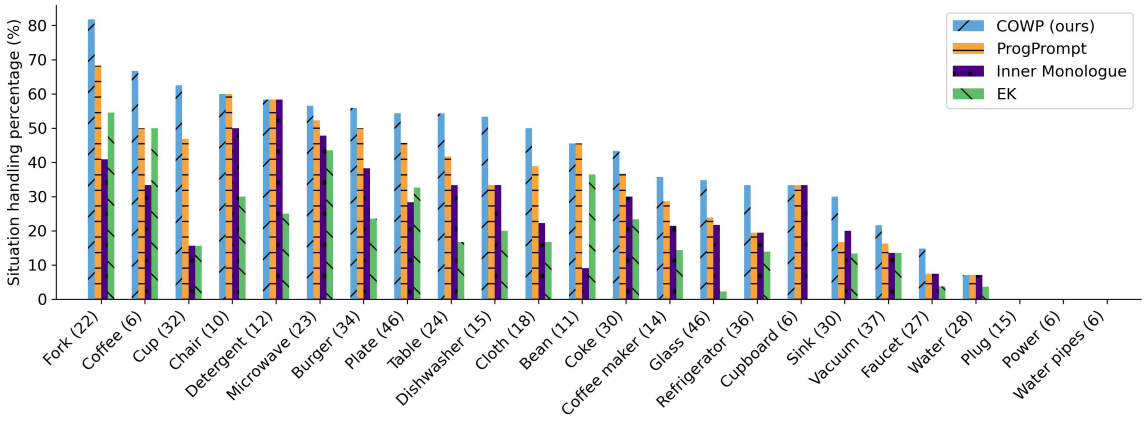


Figure 7.10: The situation handling percentages of COWP (ours) and three baseline methods under **different objects**, where the *x-axis* represents the object involved in the sampled situation, the number (X) beside each object is the occurrence of the object in situations, and the *y-axis* represents the percentage of situation handling. The objects are ranked based on the performance of COWP. In this analysis, we only display objects with an occurrence of more than 5.

indicate that COWP outperforms the baselines in all tasks. Specifically, COWP was able to handle over 45% of the situations in the first five tasks, and achieved the best performance of around 70% in the "Set table" task. There are some tasks where situation handling is more difficult for COWP. For instance, there are situations, such as "There is no water in the sink to wash the plate.", and "The robot cannot access the sink, as the door to the kitchen is locked." in task "Wash plate", where the robot could not do anything with its provided skills. By analyzing Figure 7.8 and Figure 7.9, we observe that the task "Wash plate" has a significantly higher task completion percentage than "Wash glass", despite having a similar situation handling percentage. This is due to "Wash plate" comprising only eight action sequences, compared to "Wash glass" with 12. Consequently, the robot is more prone to encountering situations in the "Wash glass" task, which hinders the robot's task completion rate.

Table 7.5: The performances of COWP (ours) and three baseline methods (ProgPrompt, Inner Monologue, and LMZSP) are compared in terms of their task completion and situation handling percentages. The comparisons are made using different LLM models (text-davinci-003, text-davinci-002, and text-ada-001) for two service tasks.

Task: Wash Glass	**LLM Models**	**COWP (ours)**	**ProgPrompt**	**Inner Monologue**	**LMZSP**
Task completion percentage (%)	text-davinci-003	**59.3**	53.3	54.7	10.7
	text-davinci-002	**53.7**	36.0	27.3	5.3
	text-ada-001	**52.0**	0.0	0.0	0.0
Situation handling percentage (%)	text-davinci-003	**18.0**	14.0	12.0	0.0
	text-davinci-002	**14.0**	6.0	4.0	0.0
	text-ada-001	**6.0**	0.0	0.0	0.0
Task: Serve Water	**LLM Models**	**COWP (ours)**	**ProgPrompt**	**Inner Monologue**	**LMZSP**
Task completion percentage (%)	text-davinci-003	**76.6**	69.3	32.7	6.7
	text-davinci-002	**73.3**	15.4	13.3	2.7
	text-ada-001	**66.0**	0.0	0.0	0.0
Situation handling percentage (%)	text-davinci-003	**34.7**	26.5	8.2	0.0
	text-davinci-002	**30.6**	8.2	6.1	0.0
	text-ada-001	**8.1**	0.0	0.0	0.0

COWP vs. Baselines by Object: Figure 7.10 compares the performance of COWP and three baselines in situation handling percentage under *different objects*. The *x-axis* represents the objects in our situation dataset and their occurrences, and the *y-axis* represents the performance of planning methods in situation handling. For instance, common situations about cup include "dirty cup," "broken cup," and "missing cup." COWP (ours) performed the best over all objects, while some baselines produced comparable performances over some objects. An important observation is that COWP produced higher success rates in situations involving "fork", "coffee", "cup", and "chair". This is because many of those relevant situations are about missing objects, and the robot can easily find their alternatives in dining domains. By comparison, those situations involving "power", "water pipes", and "plug" are more difficult to the five methods (including COWP), because addressing those situations frequently requires skills beyond the robot's capabilities.

COWP vs. Baselines under LLMs: Table 7.5 provides a comparison of COWP's performance with three baselines: ProgPrompt, Inner Monologue, and LMZSP. The evaluation specifically concentrates on the success rate of task completion and the ability to handle various situations under different LLMs. These baselines leverage LLMs for planning, while our method is classical planning augmented by LLMs. We selected three LLM models for this comparison: text-ada-001, text-davinci-002, and text-davinci-003. Text-ada–001 is considered to be the least effective, while text-davinci-003 is viewed as the most powerful [181]. From the table, it is clear that the choice of LLM has a significant impact on the baseline models' performance, with all models performing at their best under text-davinci-003. The baselines face challenges in completing task planning when utilizing the text-ada-001 model. The choice of LLM also impacts the performance of COWP, particularly when using text-ada-001. We observe that the performance of COWP under text-ada-001 becomes poor. Even when there is a situation where a robot is unable to complete a task, COWP continues with its current actions. For example, COWP will still use a dirty cup to serve water to humans. Comparing our COWP with its baselines, it is clear that the choice of LLM has a more substantial effect on the baseline models, as they heavily rely on the selected LLMs in their planning phase.

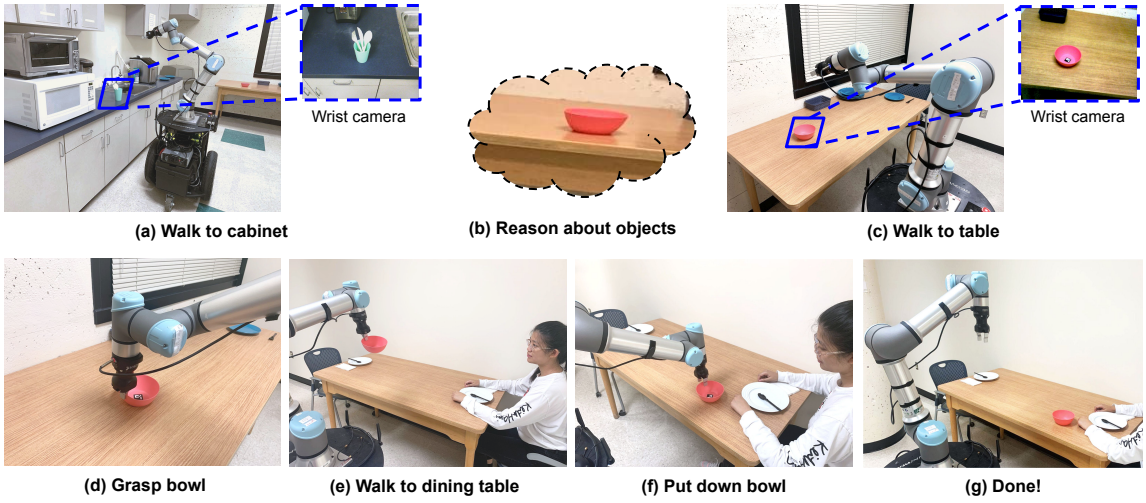


Figure 7.11: An illustrative example of COWP for open-world planning, where the robot was tasked with "delivering a cup for drinking water." **(a)** The robot walked to a cabinet, and located a cup on the cabinet. However, the robot found a situation that there were objects in the cup (a knife, a fork, and a spoon in this case). This observation was entered into the plan monitor, which queried GPT-3, and suggested that the planned action "grasp" was not applicable given the occupied cup. Accordingly, COWP updated its task planner by adding the new information that one cannot pour water into a non-empty cup. **(b)** The robot reasoned about other objects that were available in the environment, and queried GPT-3 to update the task planner about whether those objects can be used for drinking water – details in Section 7.3. It happened that the robot learned a bowl could be used for drinking water. **(c)** A new plan of delivering a bowl to the human for drinking water was generated. Following the new plan, the robot walked to the table on which a bowl was located. **(d)** The robot grasped the bowl after observing it using vision. **(e)** The robot navigated to the dining table with the bowl. **(f)** The robot put down the bowl onto the dining table, and explained that a bowl was served due to the cup being occupied, which concluded the planning and execution processes. **(g)** The task is completed.

Robot Demonstration: We demonstrated COWP using a mobile service robot that was tasked with delivering a cup for drinking water. The robot includes a UR5e arm, a Robotiq Hand-E gripper, and a Segway RMP-11 mobile base. The robot is capable of performing basic navigation and manipulation behaviors. Specifically, we employed GG-CNN [23] for object pick-and-place tasks. The navigation stack was built using the *move_base* package of the Robot Operating System (ROS) [36]. For visual scene analysis, the robot is equipped with a Robotiq Wrist Camera. With the assistance of Yolo-5 [182], the robot recognizes objects in real time. Upon receiving a top-down image, Yolo-5 generates the coordinates of the bounding boxes and identifies the objects within them, thus providing an understanding of the geometric relationships between objects. We employ a heuristics-based algorithm to determine whether the target object is occupied by another, such as when bounding boxes overlap, or if the target object is absent. While there exist more powerful tools, such as vision-language models, for visual scene analysis, computer vision is not our focus, and our goal here is to implement a basic perception component to close the perceive-reason-act loop.

Figure 7.11 shows a real-world demonstration where a robot used COWP for planning to complete the service task of "Deliver a cup for drinking water." The initial plan included the following actions for the robot:

1. Walk to a cabinet on which a cup is located,
2. Grasp the cup after locating it,
3. Walk to dining table, and
4. Put down the cup to a table where the human is seated.

However, a situation was observed after executing Action #1, and the robot found *the*

cup is occupied. The robot initially did not know whether this affects its plan execution. After querying GPT-3, the robot learned that one cannot pour water into an occupied cup, which renders its current plan infeasible. COWP enabled the robot to reason about other objects of the environment. The robot iteratively queried GPT-3 about whether X can be used for drinking water (using Prompt Template 2 in Section 7.3), where X is one of the objects from the environment. It happened that the robot learned "bowl" can be used for drinking water. With this newly learned, task-oriented commonsense knowledge, COWP successfully helped the robot generate a new plan, which used the bowl instead, to fulfill the service request. We have generated a demo video that has been uploaded as part of the supplementary materials.

8 Conclusion and Future Work

This thesis presents five efficient, feasible, and adaptable task and motion planning algorithms for robots in dynamic open-world environments. Each is summarized below, along with their prospective future work.

In Chapter 3, focusing on urban driving scenarios, we develop a safety evaluation algorithm, and a task-motion planning algorithm, called TMPUD, for autonomous driving. TMPUD, for the first time, bridges the gap between task planning and motion planning in autonomous driving. We have extensively evaluated TMPUD using a 3D urban driving simulator (CARLA) and an abstract simulator. Results suggest that TMPUD improves the task-completion efficiency in different road conditions, while ensuring the safety of driving behaviors.

In the future, we will implement TMPUD using different task and motion planners, and evaluate their performances in different testing platforms (e.g., using simulators with a physics engine) under different conditions. Also, there is the possibility of implementing and evaluating TMPUD using indoor mobile robots.

In In Chapter 4, focusing on urban driving scenarios, we develop a vision-based safety estimator and a grounded layered planning algorithm, called GLAD. Different from existing urban driving methods, GLAD considers user preference, road safety, and task-

completion efficiency – all at the same time. We have extensively evaluated GLAD using a 3D urban driving simulator (CARLA). Results suggest that GLAD produced the best performance in overall utility, while maximizing task-completion efficiency, satisfying user preferences, and ensuring the safety of driving behaviors.

There is room to improve the vision-based safety estimator. For instance, one can explore other encoders that are more effective in capturing abstract visual features. There are extreme conditions such as severe weather, car accidents, and road rage behaviors of surrounding vehicles that are not considered in our evaluations. The performance of GLAD in those challenging road conditions remains open. The above-mentioned questions can potentially lead to interesting future research.

In Chapter 5, we develop a grounded task-motion planning algorithm TMOC that can ground objects (including their physical properties) for robot planning in mobile manipulation tasks that involve complex multi-object dynamic interactions (e.g., building blocks). We assume the availability of a high-fidelity simulator, and the unavailability of accurate properties of those objects (e.g., size, weight, and shape). The robot needs to plan high-level behaviors to fulfill complex goals that require picking, moving, and placing the objects while minimizing overall action costs. Thus, we develop the TMOC algorithm to learn the object properties, state mapping function and transition function. Results from both the simulation and the real world demonstrate that TMOC improves the task-completion effectiveness and efficiency in terms of utility value and learning rate.

The work assumes that all objects are relevant to the current task, and hence TMOC grounds all of them in physics-based simulation. In the future, we plan to enable the robot to learn what objects should be grounded and when. Objects have many different physical

properties, and estimating their values requires different perception methods. This work considered size and weight, and other properties can be estimated in future work. Another direction for future research is to further increase the number of particles (each corresponding to a grounded world) for more accurately estimating object properties, where if needed, distributed computational platforms can be used. We are also interested in applying TMOC to domains beyond "stack and push" in the future. Further, our real-robot experiment can be improved by including other everyday objects with diverse properties. Finally, it is important to look into the theoretical properties of TMOC, such as its completeness and scalability. Such theoretical analysis can be difficult due to the stochastic nature of TMOC (e.g., many grounded worlds) and the iterative learning process, and we leave it to future work.

In Chapter 6, we propose LLM-GROP, which demonstrates how we can extract semantic information from LLMs and use it as a way to make commonsense, semantically valid decisions about object placements as a part of a task and motion planner - letting us execute multi-step tasks in complex environments in response to natural-language commands. In the future, we may take more information from methods like M0M [183], in order to perform grasping and manipulation of fully unknown objects in unknown scenes, and expand to a wider set of placement problems.

In Chapter 7, we develop a Large Language Model-based open-world task planning system for robots, called COWP, towards robust task planning and situation handling in open worlds. The novelty of COWP points to the integration of a classical, knowledge-based task planning system, and a pretrained language model for commonsense knowledge acquisition. The marriage of the two enables COWP to ground domain-independent commonsense

knowledge to specific task planning problems. To evaluate COWP systematically, we collected a situation dataset that includes 1,085 execution-time situations in a dining domain. Experimental results suggest that COWP performed better than existing task planners developed for closed-world and open-world scenarios. We also provided a demonstration of COWP using a mobile manipulator working on delivery tasks, which provides a reference to COWP practitioners for real-world applications.

We recognize the merits and limitations of "pure" LLM for planning (e.g., ProgPrompt) and classical planning augmented with LLMs, particularly in terms of logical consistency and flexibility of representation. Pure LLM planning has the advantage of greater flexibility, allowing it to handle a wider range of unforeseen situations, but it may be more prone to logical errors. In contrast, LLM-augmented classical planning, by parsing the LLM outputs into structured formats using hand-crafted rules, can reduce the occurrence of logical errors but may be less flexible when dealing with unforeseen situations. For instance, there might be situations where our hand-crafted rules fail to parse the output from LLMs.

It is evident that prompt design significantly affects the performance of Large Language Models (LLMs) [184], which has motivated the recent research on prompt engineering. For instance, we tried replacing "suitable" with "possible" and "recommended," in the prompt templates and observed a decline in the system performance. Researchers can improve the prompt design of COWP in future work. There is the potential that other LLMs, such as ChatGPT [14] and Bard [167], can produce better performance in open-world planning, and their performances might be domain-dependent, which can lead to very interesting future research. We present a complete implementation of COWP on a real robot, while acknowledging that there are many ways to improve the implementations.

For instance, one can use a more advanced visual scene analysis tool to generate more informative observations for situation detection, or equip the robot with more skills (such as wiping a table, moving a chair, and opening a door) to deal with situations that cannot be handled now.

We acknowledge the limitations of our method in comprehensively understanding the world state in real-world systems. To effectively monitor plans, an autonomous system requires a general-purpose perception system capable of recognizing various situations, such as object states (dirty, broken, spilled, etc.) and event occurrences. While this task is challenging, recent developments in perception, such as vision-language models, hold promise in combining visual perception with language understanding to enhance autonomous systems' capability in detecting and interpreting situations [185, 186, 187]. We are optimistic about overcoming these challenges in our future work.

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] Alper Aydemir et al. “Active visual object search in unknown environments using uncertain semantics”. In: *IEEE Transactions on Robotics* 29.4 (2013), pp. 986–1002.

[3] Marc Hanheide et al. “Robot task planning and explanation in open and uncertain worlds”. In: *Artificial Intelligence* 247 (2017), pp. 119–150.

[4] John McCarthy. *Situations, actions, and causal laws*. Tech. rep. Stanford University Technical Report, 1963.

[5] 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).

[6] Raymond Reiter. “On closed world data bases”. In: *Readings in artificial intelligence*. Elsevier, 1981, pp. 119–140.

[7] Yuqian Jiang et al. “Open-world reasoning for service robots”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 725–733.

[8] Yaqi Xie et al. “Translating natural language to planning goals with large-language models”. In: *arXiv preprint arXiv:2302.05128* (2023).

[9] Katherine M Collins et al. “Structured, flexible, and robust: benchmarking and improving large language models towards more human-like behavior in out-of-distribution reasoning tasks”. In: *arXiv preprint arXiv:2205.05718* (2022).

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

[11] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Conference on Robot Learning*. 2017, pp. 1–16.

[12] Chao Chen, Markus Rickert, and Alois Knoll. “Combining task and motion planning for intersection assistance systems”. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2016, pp. 1242–1247.

[13] 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.

[14] OpenAI. *ChatGPT*. Accessed: 2023-02-08. cit. on pp. 1, 16. 2023. URL: `https://openai.com/blog/chatgpt/`.

[15] Xavier Puig et al. “Virtualhome: Simulating household activities via programs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8494–8502.

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

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

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

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

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

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

[22] 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.

[23] Douglas Morrison, Peter Corke, and Jürgen Leitner. “Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach”. In: *arXiv preprint arXiv:1804.05172* (2018).

[24] Jeffrey Mahler et al. “Learning ambidextrous robot grasping policies”. In: *Science Robotics* 4.26 (2019), eaau4984.

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

[26] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[27] Pengfei Liu et al. “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing”. In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35.

[28] Wenlong Huang et al. “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents”. In: *Thirty-ninth International Conference on Machine Learning* (2022).

[29] Anthony Brohan et al. “Do as i can, not as i say: Grounding language in robotic affordances”. In: *Conference on Robot Learning*. 2023, pp. 287–318.

[30] Wenlong Huang et al. “Inner Monologue: Embodied Reasoning through Planning with Language Models”. In: *arXiv preprint arXiv:2207.05608*. 2022.

[31] Ishika Singh et al. “Progprompt: Generating situated robot task plans using large language models”. In: *International Conference on Robotics and Automation (ICRA)* (2023).

[32] Yafei Hu et al. “Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis”. In: *arXiv preprint arXiv:2312.08782* (2023).

[33] Tom Silver et al. “Generalized Planning in PDDL Domains with Pretrained Large Language Models”. In: *arXiv preprint arXiv:2305.11014* (2023).

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

[35] Zirui Zhao, Wee Sun Lee, and David Hsu. “Large Language Models as Commonsense Knowledge for Large-Scale Task Planning”. In: *arXiv preprint arXiv:2305.14078* (2023).

[36] 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.

[37] 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.

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

[39] 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.

[40] Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. “The social dilemma of autonomous vehicles”. In: *Science* 352.6293 (2016), pp. 1573–1576.

[41] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3354–3361.

[42] Markus Maurer et al. “Autonomous driving”. In: *Berlin, Germany: Springer Berlin Heidelberg* 10 (2016), pp. 978–3.

[43] Chana J Haboucha, Robert Ishaq, and Yoram Shiftan. “User preferences regarding autonomous vehicles”. In: *Transportation Research Part C: Emerging Technologies* 78 (2017), pp. 37–49.

[44] Philip Koopman and Michael Wagner. “Autonomous vehicle safety: An interdisciplinary challenge”. In: *IEEE Intelligent Transportation Systems Magazine* 9.1 (2017), pp. 90–96.

[45] Peng Cao et al. “Analysing driving efficiency of mandatory lane change decision for autonomous vehicles”. In: *IET Intelligent Transport Systems* 13.3 (2018), pp. 506–514.

[46] Brian Paden et al. “A survey of motion planning and control techniques for self-driving urban vehicles”. In: *IEEE Transactions on intelligent vehicles* 1.1 (2016), pp. 33–55.

[47] 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)*. 2014.

[48] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. “Learning to guide task and motion planning using score-space representation”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[49] 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.

[50] Shih-Yun Lo, Shiqi Zhang, and Peter Stone. “PETLON: planning efficiently for task-level-optimal navigation”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, pp. 220–228.

[51] Chao Chen et al. “Task planning for highly automated driving”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 940–945.

[52] Changliu Liu and Masayoshi Tomizuka. “Control in a safe set: Addressing safety in human-robot interactions”. In: *ASME 2014 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection. 2014.

[53] Changliu Liu and Masayoshi Tomizuka. “Safe exploration: Addressing various uncertainty levels in human robot interactions”. In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 465–470.

[54] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. “Model-free deep reinforcement learning for urban autonomous driving”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 2765–2771.

[55] Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. “Interpretable End-to-end Urban Autonomous Driving with Latent Deep Reinforcement Learning”. In: *arXiv preprint arXiv:2001.08726* (2020).

[56] Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. “Autonomous driving motion planning with constrained iterative LQR”. In: *IEEE Transactions on Intelligent Vehicles* 4.2 (2019), pp. 244–254.

[57] Changliu Liu and Masayoshi Tomizuka. “Enabling safe freeway driving for automated vehicles”. In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 3461–3467.

[58] Jing Bi et al. “Learning from Interventions using Hierarchical Policies for Safe Learning”. In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*. 2020.

[59] Chris Paxton et al. “Combining neural networks and tree search for task and motion planning in challenging environments”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6059–6066.

[60] Kyeong Bin Lim et al. “Behavior planning of an unmanned ground vehicle with actively articulated suspension to negotiate geometric obstacles”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 821–826.

[61] CaiJing Xiu and Hui Chen. “A behavior-based path planning for autonomous vehicle”. In: *International Conference on Intelligent Robotics and Applications*. Springer. 2010, pp. 1–9.

[62] Junqing Wei et al. “A behavioral planning framework for autonomous driving”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE. 2014, pp. 458–464.

[63] Stéphane Cambon, Rachid Alami, and Fabien Gravot. “A hybrid approach to intricate motion, manipulation and task planning”. In: *The International Journal of Robotics Research* 28.1 (2009), pp. 104–126.

[64] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. “Combined task and motion planning for mobile manipulation”. In: *Twentieth International Conference on Automated Planning and Scheduling*. 2010.

[65] Dana S Nau et al. “SHOP2: An HTN planning system”. In: *Journal of artificial intelligence research* 20 (2003), pp. 379–404.

[66] 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.

[67] Adam Houenou et al. “Vehicle trajectory prediction based on motion model and maneuver recognition”. In: *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2013, pp. 4363–4369.

[68] Samer Ammoun and Fawzi Nashashibi. “Real time trajectory prediction for collision risk estimation between vehicles”. In: *2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing*. IEEE. 2009, pp. 417–422.

[69] Saeid Amiri et al. “Augmenting knowledge through statistical, goal-oriented human-robot dialog”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 744–750.

[70] 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.

[71] Mümin Tolga Emirler et al. “Robust PID steering control in Parameter Space for highly automated driving”. In: *International Journal of Vehicular Technology* (2014).

[72] Shital Shah et al. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[73] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The 2005 DARPA grand challenge: the great robot race*. Vol. 36. Springer, 2007.

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

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

[76] Patrik Haslum et al. "An introduction to the planning domain definition language". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13.2 (2019), pp. 1–187.

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

[78] Steven M LaValle and James J Kuffner. "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and Computational Robotics* (2001), pp. 303–307.

[79] Xuemin Hu et al. "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles". In: *Mechanical systems and signal processing* 100 (2018), pp. 482–500.

[80] Wonteak Lim et al. "Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method". In: *IEEE Transactions on Intelligent Transportation Systems* 19.2 (2018), pp. 613–626.

[81] Yuqing Guo et al. "Trajectory planning for an autonomous vehicle in spatially constrained environments". In: *IEEE Transactions on Intelligent Transportation Systems* 23.10 (2022), pp. 18326–18336.

[82] Xin Huang et al. "Online risk-bounded motion planning for autonomous vehicles in dynamic environments". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 214–222.

[83] Yifan Zhang et al. "A novel learning framework for sampling-based motion planning in autonomous driving". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 1202–1209.

[84] Yifan Zhang et al. "Integrating Algorithmic Sampling-Based Motion Planning with Learning in Autonomous Driving". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 13.3 (2022), pp. 1–27.

[85] Peng Hang et al. "Human-like decision making for autonomous driving: A non-cooperative game theoretic approach". In: *IEEE Transactions on Intelligent Transportation Systems* 22.4 (2020), pp. 2076–2087.

[86] Abu Jafar Md Muzahid et al. "Optimal safety planning and driving decision-making for multiple autonomous vehicles: A learning based approach". In: *2021 Emerging Technology in Computing, Communication and Electronics (ETCCE)*. IEEE. 2021, pp. 1–6.

[87] Yongli Chen et al. "Interaction-Aware Decision Making for Autonomous Vehicles". In: *IEEE Transactions on Transportation Electrification* (2023).

[88] Yuning Wang et al. “Decision-Making Driven by Driver Intelligence and Environment Reasoning for High-Level Autonomous Vehicles: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* (2023).

[89] 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.

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

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

[92] 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)*. 2020.

[93] 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.

[94] Chongjie Zhang and Julie A Shah. “Co-optimizing task and motion planning”. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2016.

[95] 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.

[96] Xiaohan Zhang et al. “Visually grounded task and motion planning for mobile manipulation”. In: *arXiv preprint arXiv:2202.10667* (2022).

[97] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. “End-to-end model-free reinforcement learning for urban driving using implicit affordances”. In: *IEEE conference on computer vision and pattern recognition*. 2020.

[98] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[99] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[100] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[101] *CARLA Autonomous Driving Challenge*. URL: `https://leaderboard.carla.org/challenge/`.

[102] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

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

[104] 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.

[105] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "Sampling-based methods for factored task and motion planning". In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1796–1825.

[106] 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.

[107] Christian Dornhege et al. "Semantic attachments for domain-independent planning systems". In: *International Conference on Automated Planning and Scheduling*. 2009.

[108] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical planning in the now". In: *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[109] Yue Wang et al. "Task and motion policy synthesis as liveness games". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. 2016.

[110] 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.

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

[112] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *European Conference on Artificial Life*. Springer. 1995, pp. 704–720.

[113] Jie Tan et al. "Sim-to-real: Learning agile locomotion for quadruped robots". In: *arXiv preprint arXiv:1804.10332* (2018).

[114] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[115] Artem Molchanov et al. "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 59–66.

[116] Wenhao Yu et al. "Preparing for the unknown: Learning a universal policy with online system identification". In: *arXiv preprint arXiv:1702.02453* (2017).

[117] Lerrel Pinto et al. “Robust adversarial reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2817–2826.

[118] Alon Farchy et al. “Humanoid robots learning to walk faster: From the real world to simulation and back”. In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013, pp. 39–46.

[119] Yevgen Chebotar et al. “Closing the sim-to-real loop: Adapting simulation randomization with real world experience”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8973–8979.

[120] Shaojun Zhu et al. “Fast model identification via physics engines for data-efficient policy search”. In: *arXiv preprint arXiv:1710.08893* (2017).

[121] Rae Jeong et al. “Self-supervised sim-to-real adaptation for visual robotic manipulation”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2718–2724.

[122] 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.

[123] Jacky Liang, Saumya Saxena, and Oliver Kroemer. “Learning Active Task-Oriented Exploration Policies for Bridging the Sim-to-Real Gap”. In: *arXiv preprint arXiv:2006.01952* (2020).

[124] Ronen I Brafman and Moshe Tennenholtz. “R-max-a general polynomial time algorithm for near-optimal reinforcement learning”. In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[125] Neil J Gordon, David J Salmond, and Adrian FM Smith. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: *IEE Proceedings F-radar and signal processing*. Vol. 140. 2. 1993, pp. 107–113.

[126] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[127] John Hunt. “Introduction to Games Programming”. In: *Advanced Guide to Python 3 Programming*. Springer, 2019, pp. 121–123.

[128] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: (2016).

[129] Andrew Szot et al. *Habitat Rearrangement Challenge 2022*. `https://aihabitat.org/challenge/rearrange_2022`. 2022.

[130] Luca Weihs et al. “Visual Room Rearrangement”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[131] Walter Goodwin et al. “Semantically grounded object matching for robust robotic scene rearrangement”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 11138–11144.

[132] Weiyu Liu et al. “Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6322–6329.

[133] Qiuhong Anna Wei et al. “LEGO-Net: Learning Regular Rearrangements of Objects in Rooms”. In: *arXiv preprint arXiv:2301.09629* (2023).

[134] Eric Huang, Zhenzhong Jia, and Matthew T Mason. “Large-scale multi-object rearrangement”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 211–218.

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

[136] Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. “Rearrangement planning using object-centric and robot-centric action spaces”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3940–3947.

[137] Sang Hun Cheong et al. “Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 7791–7797.

[138] Vasileios Vasilopoulos et al. “Reactive planning for mobile manipulation tasks in unexplored semantic environments”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6385–6392.

[139] Jacky Liang et al. “Code as policies: Language model programs for embodied control”. In: *arXiv preprint arXiv:2209.07753* (2022).

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

[141] Pengfei Liu et al. “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing”. In: *arXiv preprint arXiv:2107.13586* (2021).

[142] Weiyu Liu et al. “StructDiffusion: Object-centric diffusion for semantic rearrangement of novel objects”. In: *arXiv preprint arXiv:2211.04604* (2022).

[143] Mohit Shridhar et al. “Alfred: A benchmark for interpreting grounded instructions for everyday tasks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10740–10749.

[144] Valts Blukis et al. “A persistent spatial semantic representation for high-level natural language instruction execution”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 706–717.

[145] So Yeon Min et al. “Film: Following instructions in language with modular methods”. In: *arXiv preprint arXiv:2110.07342* (2021).

[146] Yuki Inoue and Hiroki Ohashi. “Prompter: Utilizing Large Language Model Prompting for a Data Efficient Embodied Instruction Following”. In: *arXiv preprint arXiv:2211.03267* (2022).

[147] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. "DALL-E-Bot: Introducing Web-Scale Diffusion Models to Robotics". In: *arXiv preprint arXiv:2210.02438* (2022).

[148] Weiyu Liu et al. "Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects". In: *arXiv preprint arXiv:2110.10189* (2021).

[149] Aditya Ramesh et al. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022).

[150] Mark Chen et al. "Evaluating large language models trained on code". In: *arXiv preprint arXiv:2107.03374* (2021).

[151] Susan Zhang et al. "OPT: Open Pre-trained Transformer Language Models". In: *arXiv preprint arXiv:2205.01068* (2022).

[152] Yash Kant et al. "Housekeep: Tidying Virtual Households using Commonsense Reasoning". In: *arXiv preprint arXiv:2205.10712* (2022).

[153] Jimmy Wu et al. "Tidybot: Personalized robot assistance with large language models". In: *arXiv preprint arXiv:2305.05658* (2023).

[154] Krishan Rana et al. "SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning". In: *arXiv preprint arXiv:2307.06135* (2023).

[155] Michael Ahn et al. "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691* (2022).

[156] Ishika Singh et al. "Progprompt: Generating situated robot task plans using large language models". In: *arXiv preprint arXiv:2209.11302* (2022).

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

[158] Martin Gebser et al. "A user's guide to gringo, clasp, clingo, and iclingo". In: (2008).

[159] Valérie Boor, Mark H Overmars, and A Frank Van Der Stappen. "The Gaussian sampling strategy for probabilistic roadmap planners". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 2. IEEE. 1999, pp. 1018–1023.

[160] Walter R Gilks and Pascal Wild. "Adaptive rejection sampling for Gibbs sampling". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 41.2 (1992), pp. 337–348.

[161] Craig A Knoblock, Josh D Tenenberg, and Qiang Yang. "Characterizing abstraction hierarchies for planning". In: *Proceedings of the ninth National conference on Artificial intelligence-Volume 2*. 1991, pp. 692–697.

[162] Jörg Hoffmann. "FF: The fast-forward planning system". In: *AI magazine* 22.3 (2001), pp. 57–57.

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

[164] Sonia Chernova et al. “Situated bayesian reasoning framework for robots operating in diverse everyday environments”. In: *Robotics Research*. Springer, 2020, pp. 353–369.

[165] Vittorio Perera et al. “Learning task knowledge from dialog and web access”. In: *Robotics* 4.2 (2015), pp. 223–252.

[166] Mycal Tucker et al. “Learning unknown groundings for natural language interaction with mobile robots”. In: *Robotics Research*. Springer, 2020, pp. 317–333.

[167] Google. *Bard FAQ*. `https://bard.google.com/faq`. Accessed on April 7, 2023. 2023.

[168] David Elsweiler, Hanna Hauptmann, and Christoph Trattner. “Food recommender systems”. In: *Recommender Systems Handbook*. Springer, 2022, pp. 871–925.

[169] Ernest Davis and Gary Marcus. “Commonsense reasoning and commonsense knowledge in artificial intelligence”. In: *Communications of the ACM* 58.9 (2015), pp. 92–103.

[170] Wenlong Huang et al. “Grounded Decoding: Guiding Text Generation with Grounded Models for Robot Control”. In: *arXiv preprint arXiv:2303.00855* (2023).

[171] Cipriano Galindo et al. “Robot task planning using semantic maps”. In: *Robotics and autonomous systems* 56.11 (2008), pp. 955–966.

[172] Karthik Valmeekam et al. “Large Language Models Still Can’t Plan (A Benchmark for LLMs on Planning and Reasoning about Change)”. In: *arXiv preprint arXiv:2206.10498* (2022).

[173] Karthik Valmeekam et al. “On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark)”. In: *arXiv preprint arXiv:2302.06706* (2023).

[174] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: `2303.08774 [cs.CL]`.

[175] Cunxiang Wang, Pai Liu, and Yue Zhang. “Can Generative Pre-trained Language Models Serve As Knowledge Bases for Closed-book QA?” In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. 2021, pp. 3241–3251.

[176] Shuang Li et al. “Pre-trained language models for interactive decision-making”. In: *Advances in Neural Information Processing Systems* (2003).

[177] Peter West et al. “Symbolic knowledge distillation: from general language models to commonsense models”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2022).

[178] Chan Hee Song et al. “Llm-planner: Few-shot grounded planning for embodied agents with large language models”. In: *arXiv preprint arXiv:2212.04088* (2022).

[179] Kevin Lin et al. “Text2Motion: From Natural Language Instructions to Feasible Plans”. In: *arXiv preprint arXiv:2303.12153* (2023).

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

[181] OpenAI. *Models - OpenAI API*. `https://platform.openai.com/docs/models/overview`. Accessed: 2023-07-10.

[182] Glenn Jocher et al. "ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation". In: *Zenodo* (2022).

[183] Aidan Curtis et al. "Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 1940–1946.

[184] Ningyu Zhang et al. "Differentiable Prompt Makes Pre-trained Language Models Better Few-shot Learners". In: *International Conference on Learning Representations*. 2021.

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

[186] Deyao Zhu et al. "Minigpt-4: Enhancing vision-language understanding with advanced large language models". In: *arXiv preprint arXiv:2304.10592* (2023).

[187] Peng Gao et al. "Llama-adapter v2: Parameter-efficient visual instruction model". In: *arXiv preprint arXiv:2304.15010* (2023).