Long-Horizon Planning Under Uncertainty and Geometric Constraints for Mobile Manipulation by Autonomous Humanoid Robots

by

Alphonsus Antwi Adu-Bredu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in The University of Michigan
2023

Alphonsus Antwi Adu-Bredu

adubredu@umich.edu

ORCID iD: 0000-0002-0332-5519

ACKNOWLEDGEMENTS

To God: I am grateful to God for providing me with just the right amount of persistence, insight, good health and good fortune to contend with and even enjoy this challenging academic journey.

To my advisor: Speaking of good fortune, I am grateful to my advisor, Professor Chad Jenkins, for accepting me into his research group, giving me a great head start into academic research right at the beginning of my doctoral journey, teaching me what it means to a competent robotics researcher and more importantly, a good citizen of the field. Thank you Chad for all the great cookies and pizzas I ate during summer BBQs at your house. And of course, the Buffalo Wild Wings. They were the highlights of my summers at Ann Arbor.

To my thesis committee members: I am grateful to my committee members, Professor Nima Fazeli, Professor Jessy Grizzle, Professor Joyce Chai and Professor Herbert Winful for their guidance throughout the preparation of this dissertation. It is often said that scheduling thesis proposal and defense dates is arguably one of the most challenging tasks in pursuing a PhD. I am grateful that you were all understanding and patient and made the entire scheduling process as easy as winking. Thank you.

To my lab mates and Robotics Department colleagues: Thank you for all the great humor, great clamp jokes, great soccer games and banters, great conversations and great social outings.

To the Qualcomm Innovation Fellowship: I am grateful to Qualcomm Inc. and the Qualcomm Innovation Fellowship for funding my research towards the end of my PhD. I am also grateful to my mentors at Qualcomm for their insightful comments and recommendations during our regular meetings.

To the Department staff: Thank you Denise, Kimberly, and all the Robotics Department staff who guided me through navigating all the degree requirements, fellowship nominations, and all the million and one requests I badgered you at your offices for. My journey through graduate school would have been orders of magnitude more challenging without your prompt and invaluable assistance.

To my colleagues at Boston Dynamics: To Yeuhi, Robin, Mark, Twan, Scott, Patt and Ben. The three months I spent at Boston Dynamics were the most fun I have ever had working on robots. Thank you for your patience, guidance, good humor and brilliance.

To my parents: It's been said that the greatest gift any child can be given is good parenting. Thank you Mum and Dad for providing me with a peaceful and stable home to grow in, for the great moral examples to imitate and for all the good food I most-likely never showed enough appreciation for. Buffalo Wild Wings may have its moments but nothing beats the delicious and mouth-watering dishes my mum conjures in her kitchen. Thank you Dad for the great academic example you set for me. Your work ethic and passion for your research was undoubtedly a major inspiration for both my decision to pursue a PhD and the fuel that kept me energetic and excited to run Digit experiments at 2am.

To Digit 015: I may have uttered a few swear words at you when you broke down during crucial experiments nights before conference deadlines. I may also have posted video compilations of all your embarrassing falls during robot experiments on my website. But know that I sincerely appreciate that you were functional and available throughout my PhD. I hope that the work I present in this dissertation brings you honor and pride. Thank you Digit 015.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Autonomous humanoid robots have the potential to perform critical and labor-intensive tasks that could go a long way to improve upon the quality of human life. To realise this potential, an autonomous humanoid robot must be capable of planning the right set of long-horizon actions under the conditions of uncertainty and geometric constraints that characterize real-world environments.

This thesis proposes long-horizon planning approaches for humanoid robots under conditions of uncertainty and geometric constraints that are typical of real-world environments. The specific contributions of this thesis are, 1) A reactive and efficient task planning approach for planning under low-entropy conditions in the robot's belief of the state of the world, 2) A reactive and probabilistic long-horizon planning approach for long-horizon tasks under state estimation and action uncertainty and 3) An optimal long-horizon planning approach for geometrically constrained tasks based on mixed integer convex programming.

We demonstrate the effectiveness of the approaches presented in this thesis on object rearrangement and mobile manipulation tasks in a domestic environment using the Agility Robotics Digit Bipedal Humanoid Robot and evaluate the presented approaches on planning time and task success rate metrics.

CHAPTER I

Introduction

1.1 Motivation

The dream of autonomous robots doing useful work and relieving humans from the qualms of tedious chores has long existed in the collective minds across humanity [71]. Our society has long yearned for autonomous robots capable of performing a wide variety of tasks ranging from the mundane household chores and elderly care to the more critical tasks like emergency medical surgeries, fire-fighting and disaster relief.

For a robot designed to interact with the world and function in human spaces, the humanoid form-factor is arguably the ideal morphology of an autonomous robot. Having a human form allows robots to seamlessly occupy and function in human spaces, avoiding the need for robot-friendly structural alterations of social spaces. Though considered a very challenging problem due to their kinematic and dynamic complexity, the control of humanoid bipedal robots has seen significant advances in recent years both in terms of low-level control algorithms [104, 103, 39, 30, 50] and robot hardware [17, 13, 63, 81, 38] (demonstrated in Figures 1.1 and 1.2). Furthermore, robot perception systems that enable robots to sense and estimate their environments have also seen similar advances both in terms of inexpensive hardware [60] and fast and efficient algorithms [91, 111, 92, 113, 114].

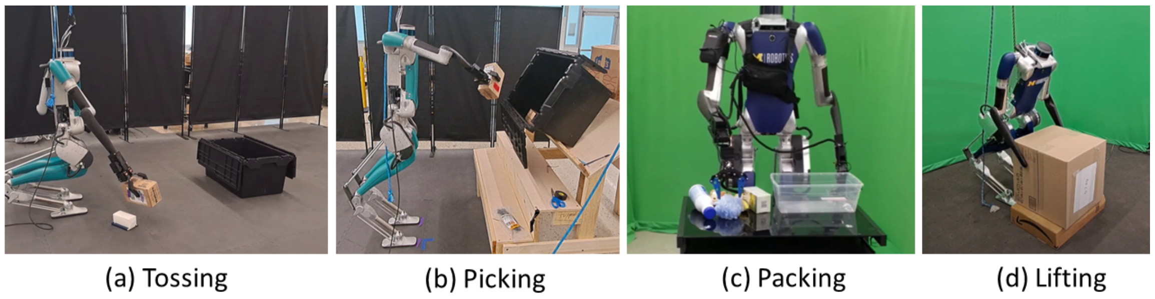


Figure 1.1: The Agility Robotics Digit Bipedal Humanoid Robot autonomously performing a number of manipulation and mobile manipulation tasks, including: (a) pick a package from the ground and toss it into the bin[4], (b) pick a package from an elevated bin [4], (c) pack assorted items into a cluttered bin [5], (d) lift a large box from the ground [4]

Given these significant advances in the low-level control and perception systems of humanoid robots, the next step on the path to complete autonomy is to equip humanoid robots with robust long-horizon planning capabilities to enable them plan and execute long-horizon tasks in the world. A unique challenge that robots face when performing long-horizon planning in real-world environments is the uncertain state estimation of their environment from their perception systems due to unavoidable factors like poor lighting conditions, occlusion or their partial knowledge of the environment. A successful task execution requires that robots account for this uncertainty when planning for long-horizon tasks. In addition to accounting for uncertainty, robots also have to account for physical geometric constraints when planning for long-horizon tasks. How far down the robot should squat in order to stably pick up a package on the ground? How high should the robot stretch in order to reach and pick a plate at the top shelf while maintaining balance? How close to a table should the robot stand in order to pick up a mug resting on the table? These questions take the form of geometric constraints that must be satisfied optimally in order for the robot to successfully plan for and execute long-horizon tasks. This thesis proposes approaches to enable autonomous humanoid robots to plan for long-horizon tasks under uncertainty and geometric constraints.

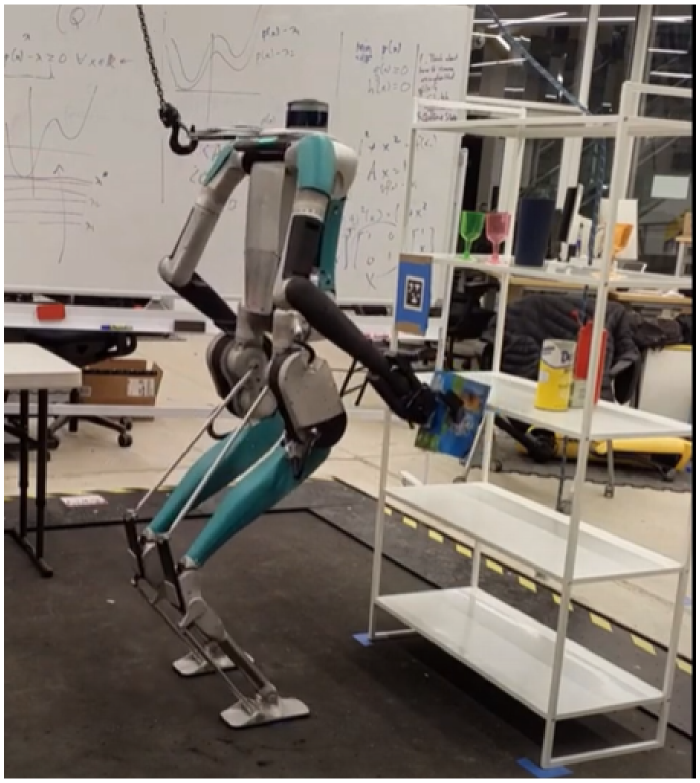


Figure 1.2: The Agility Robotics Digit Bipedal Humanoid Robot stocking a shelf with a cookie box. Figure adapted from Adu-Bredu et. al. [2]

1.2 Problem Statement

We seek to address the problem of long-horizon planning under uncertainty and geometric constraints by autonomous humanoid robots. Given a state x_t at time t, task boundary constraints C and task goals G, we seek to infer a plan π_t that satisfies constraints C and achieves goals G. The state x_t is a random variable estimated from uncertain and partial observation z_t at time t. The inferred plan π_t consists of sequential actions $a_t^1, a_t^2, \ldots, a_t^N$ where N is the length of the horizon of the plan. Each action a_t^i is made up of an action symbol σ_t^i and a set of continuous parameters ϵ_t^i needed to execute the action. This problem exhibits the Markov property because at time t, the future state x_{t+1} depends on the present state x_t but not on the past state x_{t-1}

States, actions, task goals and constraints are defined using the Hybrid Plannning

Domain Definition Language Description(HPD) described in detail in Chapter 2.1. We assume the existence of a perception system that estimates the state s_t from perceptual observation z_t. We also assume the existence of low-level controllers that can translate the current action a_t^i to a motor command u_t.

1.3 Contributions

The main contributions of this thesis are long-horizon planning approaches that account for uncertainty and geometric constraints inherent in real-world tasks.

The remainder of this thesis is structured as follows.

- Chapter II provides a literature review of long-horizon planning under uncertainty and geometric constraints. It also establishes connections between prior works and the contributions of this thesis.

- Chapter III presents a probabilistic reactive task planning approach for occasions of uncertainty in the composition of the world. This work demonstrates the relative efficiency and effectiveness of reactive planning strategies under partial observability compared to planning approaches that perform intricate probabilistic modelling and reasoning.

- Chapter IV presents a probabilistic reactive long-horizon planning approach that employs probabilistic inference on factor graphs in planning for partially observable problems. These partially observable problems exist in domains where there is uncertainty in the estimation of the state of the composition of the world.

- Chapter V presents a long-horizon planning approach that is capable of planning the optimal sequence of grounded actions to optimize a specific objective while satisfying numerical constraints. The proposed approach is able to account for geometric constraints and other continuous constraints when performing high-level symbolic task planning.

- Chapter VI presents some ideas for extensions of the work presented in this dissertation.

CHAPTER II

Background and Related Work

We desire for autonomous humanoid robots to perform useful long-horizon tasks in human environments. Long-horizon tasks are temporally extended tasks that involve taking a sequence of coherent, inter-dependent actions to achieve the task goal. A typical example of a long-horizon task is a kitchen cleaning task. To clean up your kitchen after a day of Thanksgiving dinner meal preparation, one would first load all the dirty dishes into the dish washer, wipe up the bits of food from the stove and the kitchen cabinet, sweep and mop the floor, unpack the clean dishes from the dish washer, pick up the remaining unused groceries, walk to the dish and food shelves and finally stow the dishes and groceries on the shelves. This narrated sequence of actions is a long-horizon plan a human would generate to accomplish the long-horizon task of cleaning the kitchen. Our goal is to develop long-horizon planning approaches that will enable robots to autonomously generate and execute action sequences to accomplish tasks assigned to them. This chapter is structured as follows. Section 2.1 describes the form of **inputs** to a long-horizon planning approach. It describes the various formal representations of long-horizon tasks, their unique features and the assumptions these representations make. Section 2.2 describes the form of the solution **outputs** of a long-horizon planning approach and expounds on the means by which a robot can receive and execute solution long-horizon plans. Finally Section 2.3 provides

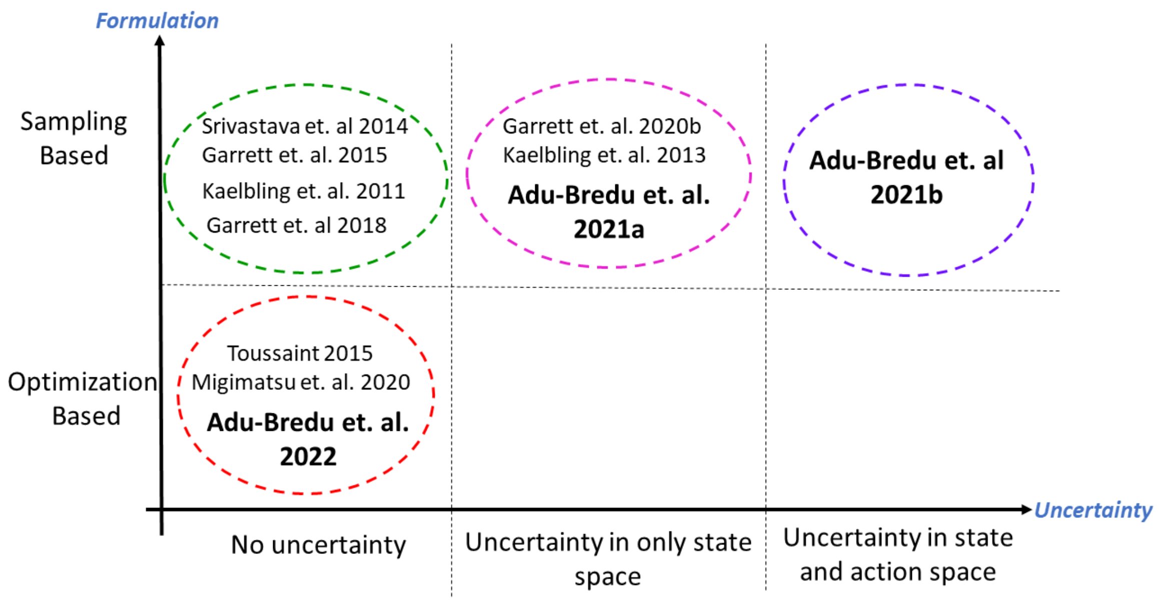


Figure 2.1: A comparison of selected long-horizon planning approaches based on their formulation (as either Sampling-Based Constraint Satisfaction or Optimization-based Constraint Satisfaction) and the extent of uncertainty they assume about their tasks. In **bold** are the contributions of this thesis

a survey of the various **approaches** for long-horizon planning, the **assumptions** they make and their **relation** to the works presented in this dissertation.

2.1 Inputs to Long-Horizon Planning

In this section, we present the various formal representations of long-horizon tasks, their features and the assumptions they make.

2.1.1 Planning Domain Description Language (PDDL)

Long-Horizon Planning problems can represented using a formal language called the Planning Domain Description Language (PDDL)[6]. The assumptions made by the PDDL representation are 1) The states of the world are finite and *fully-observable*. A state is fully-observable if the robot has absolute knowledge of all the properties of the state. 2) The actions available to the robot are instantaneous, finite and have deterministic effects. 3) The robot is the only entity in the world that can change the state of the world.

Consider the "Degas Heist" as an example of a Long-Horizon Task. This problem is made up of one agent, Robot, who looks to steal a painting by the French impressionist Edgar Degas [22] that is hanging on a wall in an art gallery. After lifting it, Robot intends to escape with the painting in his car and hide it in a storage container. The objects in this domain are `Painting` and `Bust` whilst the locations are `Gallery`, `Car`, `ParkingLot`, `Wall` and `Storage`. The state variables of the problem are

- `(holding ?obj)`, which indicates that Robot is holding the object `obj`
- `(stolen ?obj)`, which indicates that object `obj` is stolen
- `(atRob ?loc)`, which indicates that Robot is at location `loc`
- `(at ?obj ?loc)`, which indicates that object `obj` is at location `loc`

Actions are characterized by a set of *parameters*, *preconditions* and *effects*. Parameters are usually objects or locations relevant to the action's purpose. Preconditions are conjunctions and/or disjunctions of state variables of the world that have to hold true in order for the action to be executable. Effects is the resulting state after the action is executed. Actions for the "Degas Heist" domain are

```
(:action pick
:parameters (?obj ?loc)
:precondition (and (atRob ?loc) (not (stolen ?obj)) (not (holding ?obj)) (at
?obj ?loc)
:effect (and (holding ?obj) (stolen ?obj) (not (at ?obj ?loc))))


(:action place
:parameters (?obj ?loc)
:precondition (and (holding ?obj) (atRob ?loc))
:effect (and (not (holding ?x)) (at ?obj ?loc)) (not (stolen ?obj)))


(:action move
:parameters (?loc1 ?loc2)
:precondition (and (atRob ?loc1) (not (atRob ?loc2)))
:effect (and (not (atRob ?loc1)) (atRob ?loc2)))


(:action drive
:parameters (?loc1 ?loc2)
:precondition (and (atRob ?loc1) (not (atRob ?loc2)))
:effect (and (not (atRob ?loc1)) (atRob ?loc2)))
```

The initial state of the problem s_o can be

$s_o = \{$`((and (at Painting Wall) (atRob Storage) (not (stolen Painting)) (not (holding Painting)))`$\}$

and the goal state, g, can be described as

$g = \{$`((atRob storage) (at Painting Storage) (not (at Painting Wall)) (not (atRob Gallery)))`$\}$.

Given the problem description in PDDL and the initial state, the job of a task planning approach is to plan a sequence of actions that transition the state of the world from the initial state s_o to the goal state g. This problem can be reduced to a graph traversal problem and solved using graph-search algorithms like AStar [43], Breadth First Search, Depth-First Search, Goal regression, etc. [18]. The solution sequence of actions, π, also called the *plan*, for the instantiation of the "Degas Heist" problem above is

$\pi = \{$`((move Storage Gallery) (move Gallery Wall) (pick Painting Wall) (move`

`Wall Gallery) (move Gallery ParkingLot) (move ParkingLot Car) (place Painting Car) (drive ParkingLot Storage) (place Painting Storage))`}

2.1.2 Probabilistic Planning Domain Definition Language (PPDDL)

In the real world, it is often the case that certain actions have probabilistic/uncertain effects. For instance, once Robot filches the painting and is driving to the storage container location, there is a chance that he is chased by the cops and apprehended, resulting in him ending up in a Siberian prison instead of at the storage location. A language that can be used to represent such uncertainty in Long-Horizon Tasks is the Probabilistic Planning Domain Definition Language (PPDDL) [110]. The probabilistic `(drive)` action is described in PPDDL as follows;

```
(:action drive
:parameters (?loc1 ?loc2 ?loc3)
:precondition (and (atRob ?loc1) (not (atRob ?loc2)))
:effect (and (not (atRob ?loc1)) (probabilistic 0.6 (atRob ?loc2) 0.4 (atRob
?loc3))
```

The probabilistic effect reads as follows; there is a 0.6 probability at Robot ends up at `loc2`, which is the storage area and there is a 0.4 probability that the robot ends up in `loc3`, which is the Siberian prison.

2.1.3 Hybrid PDDL Description (HPD)

Hybrid PDDL Description (HPD) [2] is a novel Long-Horizon Task representation introduced in one of the contributions of this thesis. It is discussed in detail in Chapter V.

HPD is an extension of PDDL that allows for the specification of long-horizon tasks with numerical action and task constraints, numerical initial values of continuous task variables, numerical objective functions, numerical action dynamics functions, numerical preconditions and numerical effects. Similar to PDDL, an HPD description of a task planning problem is made up of two files; the `domain.hpd` file and the `problem.hpd` file.

The `domain.hpd` file describes the action primitives that the robot can execute. An action primitive has fields

- `:action` to specify the name of the action primitive
- `:parameters` to specify the symbolic and continuous parameters the action takes.
- `:precondition` to specify a conjunction of symbols whose truth-values must be true in order for the action to be executable.
- `:continuous_precondition` to specify continuous constraints on the continuous variables that must be satisfied in order for the action to be executable.
- `:dynamics` to specify dynamics functions that compute the state of the continuous variables after the action is executed.
- `:continuous_effect` to specify the numerical values of continuous variables after the action is executed.
- `:effect` to specify a conjunction of symbols that represent the state of the world after the action is executed.

The `problem.hpd` file describes the initial symbolic and continuous states as well as the goal symbolic and continuous states of the task. It also describes the task-specific constraints and the objective function to be optimized.

2.2 Outputs of Long-Horizon Planning

The output of a long-horizon approach is a grounded plan made up of a sequence of logically consistent actions, each associated with continuous parameters needed for the action to be executable. For instance, a `pick` action in a grounded plan can be associated with continuous parameters like object grasp pose, robot stance pose, etc. This grounded plan can be fed into the whole-body control architecture [61] of an autonomous humanoid robot for execution.

2.3 Overview of Long-Horizon Planning Approaches and their Assumptions

Long-horizon planning approaches can be classified under two main categories; Sampling-based methods and Optimization-Based methods. These categories can further be sub-divided based on the assumptions of uncertainty they make about the task and whether or not they can explicitly handle geometric task constraints. In the following sub-sections, we will provide an overview of Sampling-based and Optimization-based methods, their abilities to handle uncertainty and geometric constraints and their relation to the works proposed in this thesis. Figure 2.1 provides a summary of the assumptions and attributes of selected long-horizon planning approaches.

2.3.1 Sampling-based Long-Horizon Planning

In general, sampling-based long-horizon approaches interleave continuous parameter sampling routines with symbolic search over the state space. Srivastava et. al. [87] introduces an interface layer between a symbolic search routine and a sampling-based motion planning routine. This interface layer allows the sampling routine to evaluate and validate continuous action preconditions during the symbolic search pro-

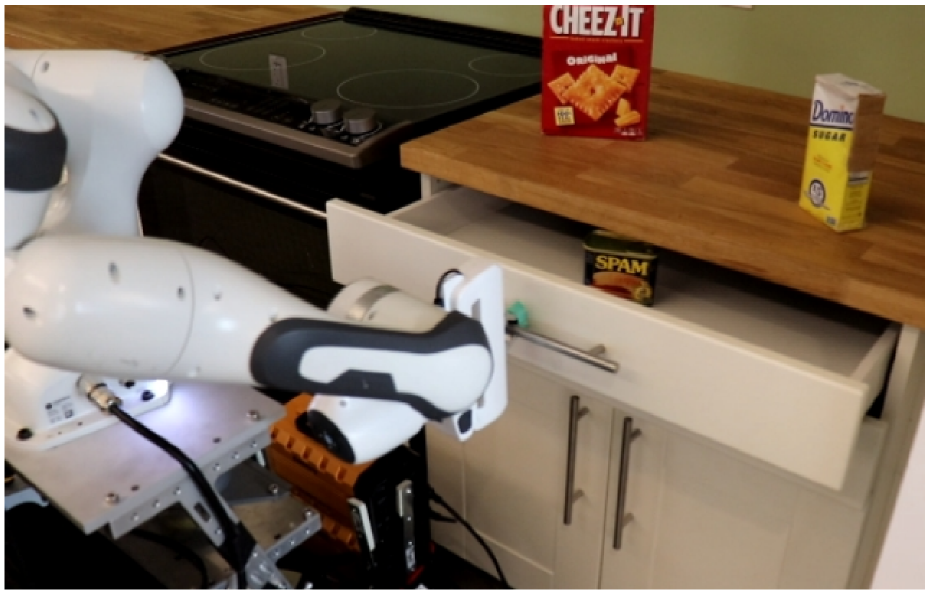


Figure 2.2: The robot pulls open a drawer to detect whether the spam object is at a continuous pose particle within the drawer. Figure adapted from Garrett et. al. [37]

cess. This interleaving between symbolic search and continuous parameter sampling ensures that output plans are physically realizable by a robot. Garrett et. al. [34, 36] devise symbolic predicates to represent geometric preconditions for actions. These symbolic predicates can then be evaluated on-demand using continuous sampling routines like sampling-based motion planning, for reachability predicates, or grasp sampling for graspability predicates.

Sampling-based long-horizon planning approaches assume that the agent has full knowledge of its deterministic domain and that the agent's actions have deterministic outcomes on its environment. These assumptions are however not representative of the kinds of domains robots operate in the real world, which are often uncertain and partially observable. There exists only a few extensions to sampling-based long-horizon planning approaches to account for the uncertainty of real-world domains. Garrett et. al. [37], depicted in Figure 2.2, introduce SS-Replan as an extension to sampling-based long-horizon planning to account for uncertainty in object pose estimation. SS-Replan represents and updates the belief over object poses using particle filtering and uses the inferred object poses to perform sampling-based long-horizon planning. SS-Replan also incorporates information-gathering actions in its plan to update its belief of the state of the world. SHY-COBRA [3], which is one of the contributions of this thesis, is a long-horizon planning approach that accounts for uncertainty in the state estimation and robot actions. SHY-COBRA takes as inputs the robot's noisy belief of the state of the world and a plan skeleton composed of symbolic actions that achieve a specified goal. SHY-COBRA then infers satisfying parameter values for the actions needed to execute the plan successfully in partially observable domains. SHY-COBRA is discussed in detail in Chapter IV. LESAMPLE [5], which is also a sampling-based long-horizon planning approach that accounts for uncertainty in the composition of the robot's environment is discussed in detail in Chapter III.

2.3.1.1 Drawbacks of Sampling-Based Long-Horizon Planning

Sampling-based long-horizon planning approaches are only able to satisfy geometric tasks constraints implicitly through the sampling of continuous variables in satisfiable regions. This satisfiability criterion only guarantees that sampled continuous variables satisfy geometric constraints. It does not guarantee optimality. This lack of optimality guarantees of sampling-based long-horizon planning approaches can often lead to the generation of feasible but sub-optimal plans. Another drawback of sampling-based long-horizon planning approaches is that, sampling as an inference strategy for constraint satisfaction is inefficient in handling geometric constraints in significantly high-dimensional spaces like the generalized coordinates of a bipedal humanoid robot. A more explicit handling of such challenging geometric constraints is needed. It is also worth noting that, due to its reliance on symbolic search methods, sampling-based long-horizon planning approaches only allow for the expression of symbolic task goals.

2.3.2 Optimization-based Long-Horizon Planning

A less-explored alternative approach for solving long-horizon tasks is to formulate them as constrained optimization problems. A constrained optimization problem takes the form

$$
\begin{aligned}
& \min_{x,u} J(x,u) \\
\text{subject to} & \\
& f(x,u) = b \\
& g(x,u) \leq c
\end{aligned}
\tag{2.1}
$$

where x and u are decision variables to be solved for, $J(x,u)$ is an objective function to be minimized and $f(x,u) = b$ and $g(x,u) \leq c$ are equality and inequality constraints that have to be satisfied by the optimal x and u solutions.

Formulating long-horizon tasks as constrained optimization problems allows for the explicit and efficient representation of geometric constraints as either equality or inequality constraints. This also allows for the global or local optimality guarantee of solution long-horizon plans. Unlike with sampling-based long-horizon planning approaches, the formulation of the long-horizon task as a constrained optimization problem allows for the expression of both symbolic and continuous task goals .

Logic Geometric Programming (LGP) [96] is an optimization-based long-horizon planning approach that seeks to solve a long-horizon task by decomposing it into two sub-problems; a symbolic planning problem specified using first-order logic and a nonlinear constrained optimization problem that operates on the continuous variable values. LGP solves the symbolic planning problem using Monte Carlo Tree Search. The solution plan is then used to constrain the nonlinear constrained optimization problem to solve for optimal continuous variable values needed to execute the plan.

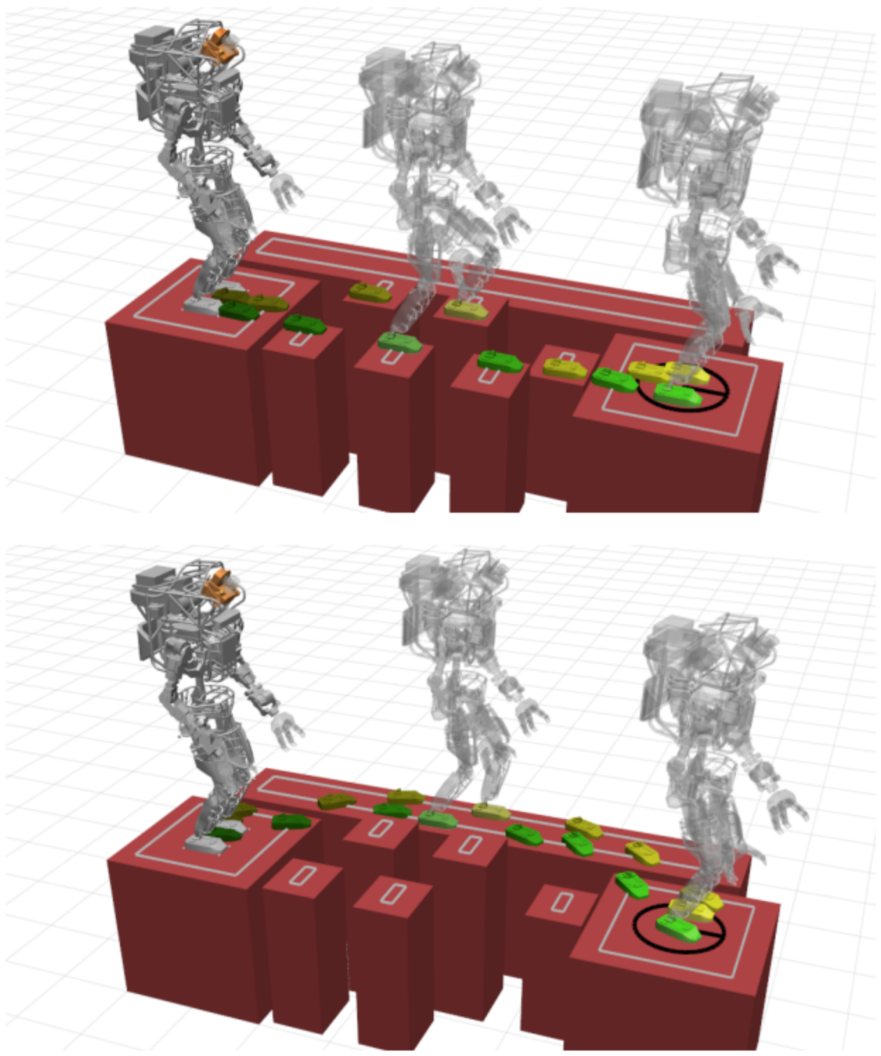
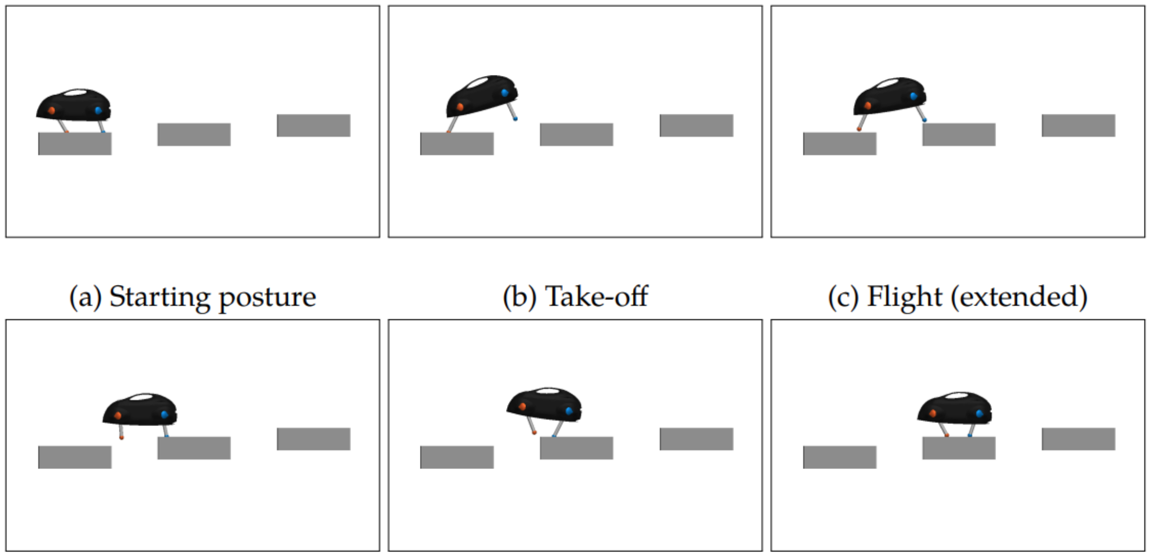


Figure 2.3: The Atlas bipedal humanoid robot planning footsteps across a set of stepping stones using a Mixed Integer Programming planner. Figure adapted from Deits and Tedrake [23]

Although LGP provides all the benefits of optimization-based long-horizon planning, the decomposition of the logical reasoning and numerical optimization into two separate problems results in an inefficient and hyper-specialized approach that is difficult to apply to new problems. A general and more efficient formulation of the problem is to encode the long-horizon planning problem as a **Mixed Integer Program**.


(a) Starting posture (b) Take-off (c) Flight (extended)

(d) Touch-down (e) Flight (gathered) (f) Touch-down

Figure 2.4: The Big Dog planar quadruped robot computes bounding motions over uneven terrain with gaps using a Mixed Integer Programming planner. Figure adapted from Valenzuela et. al. [97]

2.3.2.1 Robot Planning as Mixed Integer Programming

Mixed Integer Programs are mathematical optimization programs that have both integer- and real-valued variables. They often take the form

$$\min_{x,u} J(x,u)$$

subject to

$$C(x,u) \leq D$$

$$x \in \mathbf{Z}^m, u \in \mathbf{R}^n$$

where x is an Integer variable, u is a Real variable, J is an objective function of x and u and $C(x,u)$ is an inequality constraint function that depends on both x and u.

The ability of Mixed Integer Programs to have both Integer and Real variables makes them convenient for formulating sequential planning problems that involve taking discrete actions which are subject to continuous constraints [1, 23, 48, 97].

Mixed Integer Programs have been applied to hybrid problems like footstep planning for bipedal robots [23] as shown in Figures 2.3 and 2.4 as well as contact-implicit trajectory optimization for grasp planning [1], [48]. A Mixed Integer Program is solved using algorithms like branch-and-bound [75], cutting-plane [40] and branch-and-cut [76]. There exists excellent off-the-shelf software like Mosek [74], CPLEX [51], Juniper [54], SCIP [82] and Gurobi [41] that provide efficient implementation of these algorithms for solving Mixed Integer Programs.

GTPMIP [2] which is a contribution of this thesis, formulates the entire long-horizon task as a single **Mixed Integer Program** and solves it to optimality to output geometrically feasible and optimal long-horizon plans that can be successfully executed with a robot. GTPMIP is discussed in detail in Chapter V.

2.3.2.2 Drawbacks of Optimization-based Long-Horizon Planning

Optimization-based Long-Horizon Planning approaches like GTPMIP do not account for uncertainty in the robot's state estimates. As such, they are likely to perform poorly in situations where the robot's state estimates have high uncertainty. Re-formulating the Mixed Integer Program in GTPMIP as a Probabilistic Mixed Integer Program as proposed by Vielma et. al. [99] will enable GTPMIP to account for uncertainties in both continuous variables and in geometric constraints.

CHAPTER III

Elephants Don't Pack Groceries: Robot Task Planning for Low Entropy Belief States

Recent advances in computational perception have significantly improved the ability of autonomous robots to perform state estimation with low entropy. Such advances motivate a reconsideration of robot decision-making under uncertainty. Current approaches to solving sequential decision-making problems model states as inhabiting the extremes of the perceptual entropy spectrum. As such, these methods are either incapable of overcoming perceptual errors or asymptotically inefficient in solving problems with low perceptual entropy. With low entropy perception in mind, we aim to explore a happier medium that balances computational efficiency with the forms of uncertainty we now observe from modern robot perception. We propose an approach for efficient task planning for goal-directed robot reasoning. Our approach combines belief space representation with the fast, goal-directed features of classical planning to efficiently plan for low entropy goal-directed reasoning tasks. We compare our approach with current classical planning and belief space planning approaches by solving low entropy goal-directed grocery packing tasks in simulation. Our approach outperforms these approaches in planning time, execution time, and task success rate in our simulation experiments. We also demonstrate our approach on a real world grocery packing task with physical robot.

3.1 Introduction

Sequential decision-making problems have often been modelled as either fully-observable or partially observable. Fully observable models have no entropy in states and actions whilst partially observable models have high entropy in states and actions. With these models have come classical planning approaches [33, 56, 44, 46] for solving zero entropy problems and belief space planning approaches [85, 37, 58] for solving high entropy problems. Visualizing entropy as a spectrum, classical planning approaches plan with models on one extreme end of the entropy spectrum whilst belief space planning approaches plan with models on the other extreme. Recent advances in robot perception systems, both in terms of inexpensive hardware [60] and fast and efficient algorithms [91, 111, 92, 113, 114] have significantly reduced the state estimation entropy when used for robot manipulation. When a robot is equipped with such a low entropy perception system, the robot's sequential decision-making problem does not fall at either extremes of the entropy spectrum. The problem falls in an intermediate region on the spectrum where neither family of approaches are equipped to exploit the low entropy nature of the problem to solve it efficiently.

Classical planning approaches do not account for uncertainty so they often fail to generate feasible plans in uncertain domains. Some belief space planning approaches attempt to exactly solve for the optimal policy that maps belief states to actions [93]. Since solving for the optimal policy exactly becomes intractable for realistic problems, other belief space planning approaches approximate the belief space through sampling [85, 86]. Although these approximate methods are tractable, they tend to be inefficient for low entropy state spaces.

Results from early work in Embodied Intelligence by Brooks et. al. [11, 12] demonstrate that, methods that plan and act on loose models of the world and rely on sensor feedback to adjust their behavior are often more efficient and practical than their counterparts that perform explicit modelling of all possibilities before taking

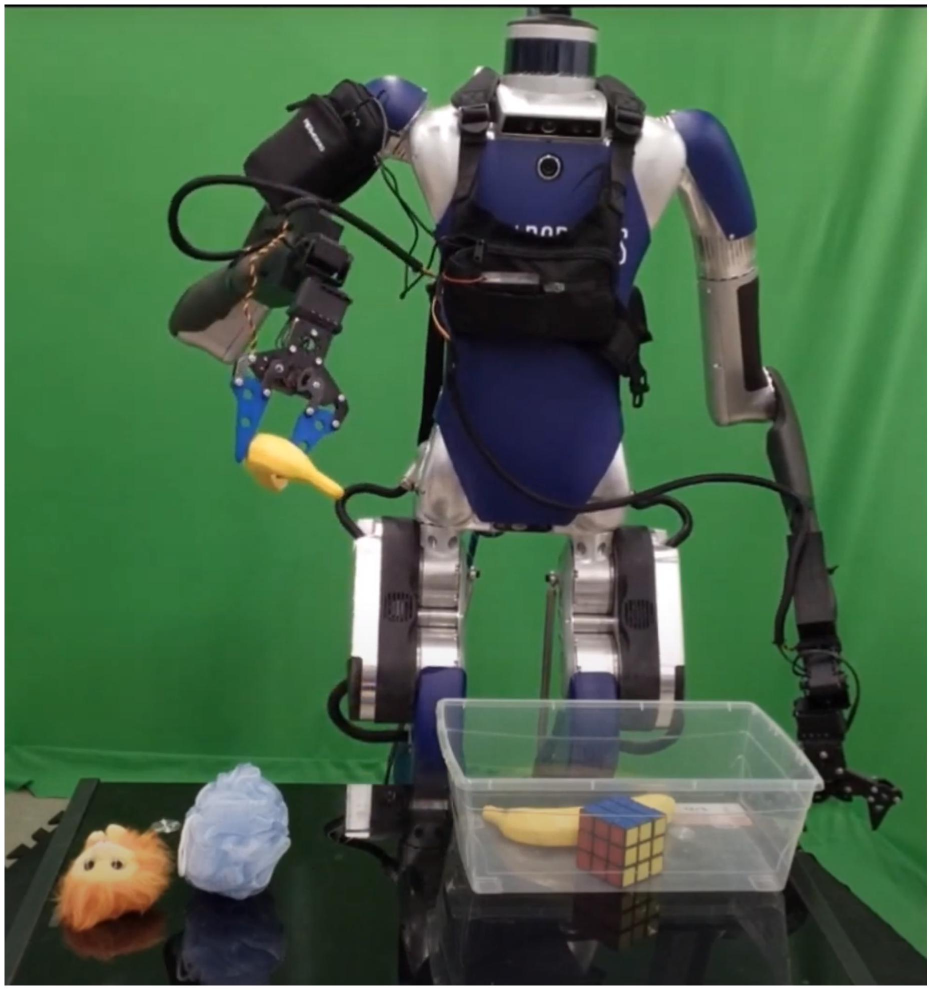


Figure 3.1: A sequential grocery packing task using LESAMPLE. Digit robot equipped with parallel grippers is able to efficiently pack the groceries under low perceptual entropy to satisfy given goal constraints.

an action. These results are also echoed in more recent work [109, 69] that show replanning approaches to be more efficient in domains with stochastic action effects than probabilistic planning approaches. Inspired by these results, we hypothesize that for a state space with low perceptual entropy, a simple replanning approach that samples from the belief space and plans using this sample will be more efficient than belief space planning in solving the task at hand.

In light of this, we propose a decoupled approach to goal-directed robotic manipulation that builds on the respective strengths of classical and belief space planning. As motivated by Sui et al. [91], task planning can be performed on state estimates from perceived belief distributions, and updated when the perceptual probability mass shifts to a different state estimate. Building on this idea and recent work in replanning algorithms [109], we propose **Low Entropy Sampling planner** (**LESAMPLE**) as a simple and efficient online task planning algorithm for solving problems with low entropy in state estimation and deterministic action effects.

The concept of replanning with estimates from the belief space is not novel and has been employed in works like Yoon et. al. [109]. This paper does not claim to propose an entirely new algorithm. We instead aim to demonstrate the efficiency benefits a simple replanning approach could have over belief space planning methods, when used to solve low entropy planning problems.

We benchmark LESAMPLE against current classical planning and belief space planning approaches by solving low entropy goal-directed grocery packing tasks in simulation as shown in Figure 3.1. LESAMPLE outperforms these approaches with respect to planning time, execution time, and task success rate in our simulation experiments.

3.2 Related Work

3.2.1 Classical Planning

Classical planning approaches [43, 18, 89] are used to solve fully-observable and deterministic problems. These approaches are fast and usually come with convergence and optimality guarantees, making them convenient to use on suitable problems. They however do not account for entropy when planning so they often generate infeasible plans in uncertain domains.

To solve problems of a sequential nature such as grocery packing, the robot has to be able to reason over both symbolic states of the world as well as continuous states. This family of problems is known as Task and Motion Planning Problems [87,

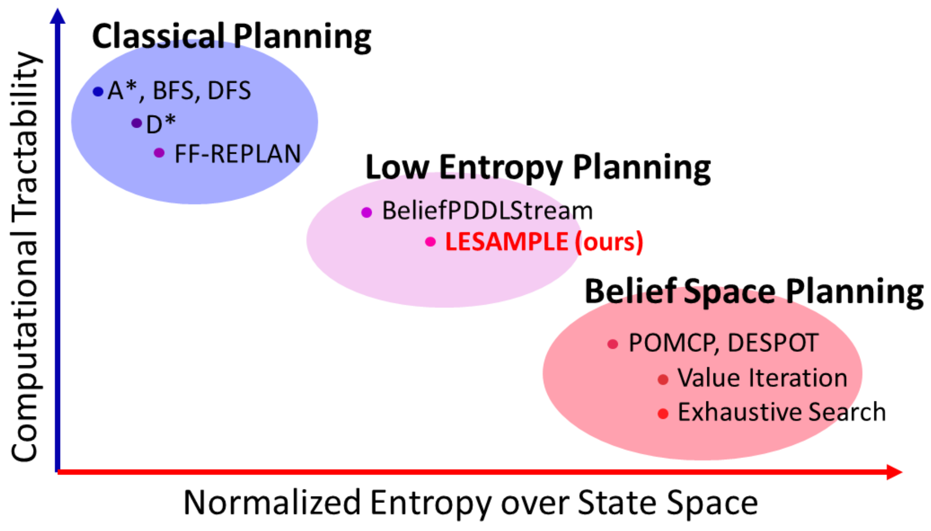


Figure 3.2: An illustration comparing the relative the computational tractability of planning algorithms and the level of perceptual entropy in problems they are designed to solve. Classical planning algorithms are computationally tractable (fast) and are designed to solve problems with no entropy in their state space. Belief space planning algorithms are generally slow on reasonably complex problems and are designed to solve problems with high perceptual entropy in their state space. Our algorithm, LESAMPLE, is designed to efficiently solve problems with low perceptual entropy.

58, 105, 33, 77, 36]. Works such as Kaelbling et. al. [56], Srivastava et. al. [87] and Garrett et. al. [36] have focused on ways to interleave the symbolic planning involved in task planning with the continuous-space planning involved in motion planning in order for the robot to generate feasible plans and actions. In our proposed LESAMPLE method, we first generate a symbolic plan and later use continuous parameter sampling and sampling based motion planning [53] to generate continuous trajectories for performing the task at hand.

3.2.2 Belief Space Planning

Sequential decision-making problems with high entropy in state estimation and action effects are often modelled as Partially Observable Markov Decision Processes (POMDP) [55]. The states in a POMDP are probability distributions called belief states. To solve a POMDP is to find an optimal policy that maps belief states to actions. However solving POMDPs exactly is intractable due to the curse of dimensionality [55] and the curse of history [79]. As such, belief space planning methods such as POMCP [85] and DESPOT [86] are used to solve POMDPs approximately. POMCP [85] uses Monte Carlo Sampling to sample the belief state and belief transitions and uses Monte Carlo Tree Search [16] to search for an optimal policy. DESPOT [86] improves upon POMCP's worst case behavior by sampling a small number of scenarios and performing search over a determinized sparse partially observable tree. Our proposed approach represents each state as a set of hypotheses which are weighted based on the robot's observation. We use weighted sampling [106] to sample from the weighted hypotheses to get a reliable estimate of the states. We evaluate LESAMPLE against POMCP and DESPOT in our experiments.

3.2.3 Integrating Belief Space Representation with Classical Planning

FF-Replan [109], a classical planning approach, attempts to solve problems with no entropy in state estimation but high entropy in action effects by constantly re-planning. FF-Replan determinizes the action effects through choosing the effect with the highest confidence. It then applies Fast-Forward [46] to plan in the determinized domain and re-plans whenever there is an inconsistency caused by the disregard of the entropy in the domain. This algorithm is shown to work quite well for certain problems and terribly for others depending on how well the determinization reflects the true action effects of the domain. Other approaches such as BeliefPDDLStream [37] use particles to represent the belief space and update the particles after each observation using a particle filter and replans using this estimate of the belief space. Our proposed approach represents each state as a set of weighted hypotheses and use weighted sampling [106] to sample from the weighted hypotheses to get a reliable estimate of the states. We then employ symbolic planning to plan in the sampled states and reweight, resample and replan when needed. Figure 3.2 shows a graphical comparison of LESAMPLE with other classical and belief space planning methods.

We also evaluate LESAMPLE against FF-Replan and BeliefPDDLStream in our experiments.

3.2.4 Embodied Intelligence

Early research in Embodied Intelligence [12, 11] demonstrated the efficiency of methods that plan and act on loose models of the world and rely on sensor feedback to adjust their behavior. These approaches worked best for domains where execution errors were reversible. More recent work in planning under uncertainty [109, 69] have also produced results that show the efficiency of replanning approaches over deliberate probabilistic planning methods in partially observable domains. These early work also gave rise to approaches [49, 15, 107] that explicitly perform information gathering actions and decide the next best action based on the obtained sensory information. LESAMPLE updates its belief after every action taken and replans whenever the updated belief doesn't match the predicted effect of the executed action.

3.2.5 Bin Packing

A vast body of work has addressed the robot bin packing problem [102, 101, 108]. Amongst these, Wang and Hauser [102], Weng et. al. [108] consider the problem from a geometric perspective and try to find the optimal packing arrangement of objects such that they use up a minimum number of bins and a minimum amount of space. Wang and Hauser [101] goes further to optimize for space-efficient packing arrangements that result in stable object piles. With the grocery packing task considered in this work, our proposed approach, LESAMPLE, does not explicitly optimize for efficient space usage when packing. We mainly just sample free placement poses in the destination bin that are large enough for the item in hand to be placed at.

3.3 Problem Formulation

The state space is represented as an object-centric scene graph. The scene graph, $\Psi(V, E)$, represents the structure of the scene. Vertices V in the scene graph represent the objects present in the scene whilst the edges E represent spatial relations between the objects.

We assume a perception system that takes robot observations and returns a belief $\mathcal{B}(\Psi)$ over scene graphs for the current scene. An example of such perception system is described in Section 3.5. Given the task goal conditions G and the current scene belief $\mathcal{B}(\Psi)$, the goal for the robot is to plan a sequence of actions $\{a_0, a_1, \dots\}$ to achieve the goal conditions G under the perceptual uncertainty, and replan when needed.

3.4 LESAMPLE

The proposed planning algorithm, LESAMPLE, takes in goal conditions, G, and the belief over the current scene graph $\mathcal{B}(\Psi)$, efficiently plans out a sequence of actions to achieve G and replans when necessary. LESAMPLE is developed to solve problems with partially observed states and deterministic action effects.

As described in Algorithm 1, LESAMPLE takes in goal conditions G and the current belief over scene graphs $\mathcal{B}(\Psi)$ as input. LESAMPLE first samples a scene graph Ψ_s from $\mathcal{B}(\Psi)$ (as described in section 3.5), and formulates Ψ_s and G as a PDDL[6] problem . LESAMPLE then uses a symbolic planner (Fast Downward [44] in our implementation) to solve the PDDL problem and generate a task plan π. After taking each action in π, the robot takes a new observation Φ. A validation function, $Validate(\Psi_s, a, \Phi)$, checks for inconsistency in the action effects. In particular, the validation returns $True$ if the new observation Φ after executing action a matches the predicted observation based on a and sampled scene graph Ψ_s, and returns $False$

otherwise. In our experiments, $Validate(.)$ checks if the object picked up by the robot matches the sampled scene graph hypothesis after every pick action. If the $Validate$ function returns $True$, robot continues to execute the next action in π. Otherwise, π is discarded, the scene belief $\mathcal{B}(\Psi)$ is updated based on the observation Φ and LESAMPLE is then called recursively with the original goal condition G and the updated $\mathcal{B}(\Psi)$ as parameters. LESAMPLE runs until either the last action in the current plan π is successfully executed or timeout is reached.

We provide details on the continuous motion parameters used in action execution in Section 3.5.4. By combining belief samples from belief space representation with the fast, goal-directed classical planning, LESAMPLE is able to efficiently plan for sequential decision making tasks with low entropy belief states.

Algorithm 1: LESAMPLE algorithm

```
Input: Goal conditions, $G$, and Belief of current scene graph, $\mathcal{B}(\Psi)$
Function LESAMPLE($G$, $\mathcal{B}(\Psi)$):
    $\Psi_s \leftarrow$ sample from $\mathcal{B}(\Psi)$
    $\pi \leftarrow FastDownward(\Psi_s, G)$
    foreach $a \in \pi$ do
        take action $a$
        take new observation $\Phi$
        $valid \leftarrow Validate(\Psi_s, a, \Phi)$
        if $valid = False$ then
            update belief $\mathcal{B}(\Psi)$ given $\Phi$
            LESAMPLE $(G, \mathcal{B}(\Psi))$
            return
        end
    end
    return
End Function
```

3.5 Implementation

3.5.1 Belief over Scene Graphs

For our experiments, we build a perception system that returns a $\mathcal{B}(\Psi)$ belief over scene graphs given robot observation. We trained a Faster R-CNN [80] detector for the 8 grocery object classes that we considered in the simulation experiments. For the ith detected object, the detector returns a confidence vector, which is then interpreted as the belief over object classes, $\mathcal{B}_i(c), c \in C$. C is the set of all possible object classes. The spatial relations between detected objects given their detected locations are deterministically derived. With the assumption that the objects are independent from one another, we approximate the belief over scene graphs $\mathcal{B}(\Psi)$ as

$$\mathcal{B}(\Psi) \propto \prod_{i}^{N} \mathcal{B}_i(c)$$

where N is the number of detected objects.

The belief over ith detected object, $\mathcal{B}_i(c)$, is approximated by a set of weighted object hypotheses $\{(o_k, p_k)|k = 1, \ldots, |C|\}$, where $o_k = (c_k, a_k, r_k)$. For each hypothesis, c_k is the object class, a_k is object attributes, r_k is its spatial relations with other objects in the scene graph, and p_k is the weight of the hypothesis, which is equivalent to the detection confidence score corresponding to c_k. The object attributes a_k (e.g. heavy or light) are deterministically associated with object class.

In order to draw one scene graph sample Ψ_s from $\mathcal{B}(\Psi)$, we individually draw one object hypothesis o_k from each $\mathcal{B}_i(c)$, such that

$$\Psi_s = \{o_k^i | i = 1, \ldots, N\}$$

where again N is the number of detected objects. We used weighted sampling [106] to sample the individual object hypothesis.

3.5.2 Grocery Packing Goal Conditions

For the grocery packing task in our experiments, for each object hypotheses $o_k = (c_k, a_k, r_k)$ in the scene graph, we consider the object to be either a heavy or light, i.e. $a_k \in \{heavy, light\}$. The goal condition for the grocery packing task, which is specified as symbolic predicates in PDDL, is to have all objects packed into the box such that, heavy grocery objects are placed at the bottom of the box, and light grocery objects are placed on top of them.

3.5.3 Action schemas

A plan is made up of a sequence of *action schemas*. An action schema consists of a set of free parameters (`:parameters`), conjunctive boolean pre-conditions (`:precondition`) that must hold for the action to be applicable and conjunctive boolean effects (`:effect`) that describe the changes in the state after the action is executed. Boolean conjunctive operators used are `or, not, and`. The pick and place action schemas used in our experiments are described below:

```
(:action pick
:parameters (?x)
:precondition (and (topfree ?x) (handempty))
:effect (and (holding ?x) (not (handempty)) (not (topfree ?x)))
)
(:action place
:parameters (?x ?y)
:precondition (and (holding ?x) (topfree ?y))
:effect (and (not (holding ?x)) (on ?x ?y) (handempty) (not (topfree ?y)) (topfree
?x))
)
```

3.5.4 Continuous Variables

In executing actions in a plan, the robot requires certain continuous values such as collision-free arm trajectories, object grasp poses and object placement poses. We use the BiRRT [53] motion planning algorithm to generate collision-free trajectories for `pick` and `place` actions. We determine the grasp pose of an object by querying its 6D pose from the simulator and computing a corresponding grasp configuration of the robot's gripper to pick the object from the top. The `place` actions specify destination surfaces on which to place the picked item. For example, the action `(place banana bowl)`, requires that the `banana` is placed on the surface of `bowl`. We query the Axis-Aligned Bounding Box (AABB) of both the item in hand (`banana`) and the destination surface (`bowl`) and sample legal placement poses on the surface of the AABB of the destination surface where we can place the item in hand. The first sampled legal placement pose is chosen as the placement pose of the object in hand.

3.6 Quantifying Entropy

The state space of all possible scene graphs is beyond tractability for modern computing. This space can be composed of all possible classes of objects, all possible number of objects, all possible enumerations of the 6D pose of objects, all possible spatial relations between objects and the attributes of objects. To be computationally tractable, we make the following assumptions to constrain the state space:

- The set of all possible object classes is finite and known.
- The perception system detects objects that are not fully occluded by other objects from the robot's field of view. Note that, the robot does not have prior knowledge of the total number of objects to expect. Thus the scene graph is made up of only detected objects and their spatial relations.

- The perception system can deterministically infer the 6D pose of detected objects. Note that, there is uncertainty in the recognition of the class of detected objects.

- The perception system can deterministically infer spatial relations between detected objects from a 3D observation. In this work, we only consider the stacking spatial relations. In the PDDL problem description, the stacking relations are represented with axiomatic assertions `(on` o_i o_j`)` for the assertion that object o_i is stacked on object o_j and `(topfree` o_i`)` for the axiomatic assertion that no objects are stacked on object o_i.

- We deterministically associate detected objects with their respective attributes (either heavy or light).

As a result, the constrained state space of scene graphs, Ψ, will include scene graphs that have the same number of vertices as the number of detected objects, with each graph consisting of all possible enumerations of the object classes.

We quantify the entropy of the belief over scene graphs $\mathcal{B}(\Psi)$ as the normalized sum of Shannon entropies [68] of the beliefs of detected objects, i.e.

$$H = -\frac{1}{H_{max}} \cdot \sum_{i=1}^{N} \sum_{c \in C} p_i^c \log_2 p_i^c \tag{3.1}$$

where C is the set of all possible object classes, N is the number of detected objects, p_i^c is the probability of class c of ith detected object in belief $\mathcal{B}_i(c)$, as explained in Section 3.5.1. H_{max} is the maximum possible entropy occurring when the belief over scene graphs is uniformly distributed. i.e.

$$H_{max} = -\sum_{i=1}^{N} \sum_{c \in C} p_i^c \log_2 p_i^c \tag{3.2}$$

where p_i^c follows a uniform distribution.

For the grocery packing task we consider in this work, we use Equation 3.1 to classify the perceptual entropy levels of grocery packing tasks. On one extreme, $H = 0$ for a scene graph estimate with no uncertainty. On the other extreme, $H = 1$ for a scene graph with high uncertainty. In our experiments, we perform grocery packing on scene graphs with H values from 0.1 to 0.9.

3.7 Experiments

We compare the performance of LESAMPLE with replanning and belief space planning methods on low entropy grocery packing tasks (H values between 0.3 and 0.5), shown in results in Table 3.1 and Figures 3.4, 3.5 and 3.6 and a broader range of entropy values (H values from 0.1 to 0.9), shown in results in Figure 3.7. The simulation environment we use is depicted in Figure 3.1. The goal condition for the grocery packing task is to have all items packed into the box such that, heavy grocery items are placed at the bottom of the box, and light grocery items are placed on top of them. Experiments were run in the Pybullet simulation [19]. We use a simulated Digit robot [81] equipped with suction grippers on both arms. 8 3D models of grocery items from the YCB dataset [14] were used as grocery items in the experiments. A Faster R-CNN [80] object detector is trained to detect these grocery items and return a confidence score vector for each detected object. We normalize these confidence scores, add entropy based on the specific H value of the task, as prescribed in Equation 3.1, and form the belief over scene graphs $\mathcal{B}(\Psi)$. The experiments were run on a laptop with 2.21GHz Intel Core i7 CPU, 32GB RAM and a GTX 1070 GPU. We also demonstrate our approach on a real world grocery packing task with a physical Digit robot. A summary of our approach as well as the real world demo can be seen in the accompanying video and at this url: https://youtu.be/im6tve9-9A0.

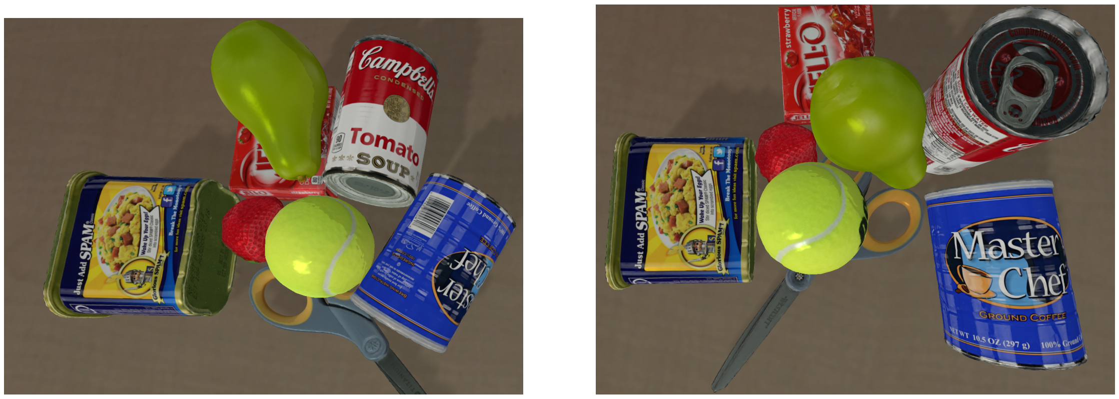
The following methods are benchmarked in this experiment:

- **FF-REPLAN**: This algorithm [109] performs symbolic planning on a determinized belief over detected objects. It determinizes the belief over scene graphs by choosing the hypothesis with the highest probability in $\mathcal{B}_i(c)$ for each detected object, i. The algorithm then formulates the determinized scene graph and the goal conditions (Section V-B) as a PDDL [6] problem and solves it using Fast Downward [44] to generate a plan to pack objects into the box. The algorithm uses the *Validate* function (same as in LESAMPLE in Algorithm 1) to check if the object picked up by the robot matches the determinized scene graph after every pick action. If *Validate* returns *True*, the next action in the plan is executed. If *Validate* returns *False*, a replan request is triggered. The belief over detected objects is updated and determinized again. A new plan is generated accordingly. The robot plans and executes until either all the objects are packed or the 15-minute timeout is reached.

- **LESAMPLE**: This algorithm performs LESAMPLE on the belief over detected objects to pack them into the box. LESAMPLE terminates either when all the objects are packed into the box or when the 15-minute timeout is reached.

- **BPSTREAM***: This algorithm is a variation of BeliefPDDLStream [37] adapted to suit our grocery packing task. We replace the streams in the original BeliefPDDLStream with simple parameter samplers for sampling motion plans and other continuous action parameters as described in Section 3.5.4. We use a set of 8 weighted particles to represent the belief of each detected object.

- **POMCP-ER**: POMCP-ER is a variation of the POMCP [85] belief space planning algorithm with episodic rewards. Here, the robot receives a reward of 10 whenever an item is packed into the container, a reward of 100 when the arrangement satisfies the packing conditions and a reward of -10 when the arrangement fails to satisfy the packing conditions. POMCP-ER is also restricted to 10 it-



((a)) ((b))

Figure 3.3: Examples of initial cluttered scenes

erations, each with a rollout depth of 10 and represents the belief set with 10 particles. Since grocery packing is a goal-directed task, we set the discount factor to 1, thus future rewards are just as valuable as immediate ones. To narrow the focus of the Monte Carlo Tree Search in POMCP, as prescribed by [85], we use domain knowledge by specifying the subset of preferred actions at each node in the search tree.

- **DESPOT**: We use an anytime and regularized version of the DESPOT belief space planning algorithm[86]. DESPOT improves upon POMCP's poor worst case behavior by sampling a small number of scenarios (3 scenarios in our implementation) and searching over a determinized sparse partially observable tree. Here, we use the same reward function, maximum number of iterations, maximum rollout depth and discount factor as POMCP-ER.

The methods are benchmarked on low-entropy Grocery Packing tasks. Their performance results are displayed in Table 3.1 and Figures 3.4, 3.5 and 3.6.

We run each planning method on 5 different initial arrangements of the grocery items, examples of which are showin in Figure 3.3. In Figure 3.4, we show planning time and execution time averaged across 5 initial scenes. For each initial scene, we run each method 5 times.

As shown in Table 3.1 and Figures 3.4, 3.5 and 3.6, across all tasks in our experiments, LESAMPLE outperforms other baseline methods by having the least completion times, making the least number of mistakes and as such requiring the least number of pick-and-place actions to complete the task. BPSTREAM* has a slightly higher execution time than LESAMPLE and a significantly higher planning time than both FF-REPLAN and LESAMPLE. The belief space planning algorithms, DESPOT and POMCP-ER, have the worst results for every metric. DESPOT and POMCP-ER make fewer number of mistakes because they spend majority of the time planning and are only able to take a few actions before the 15 minute timeout is reached. DESPOT however performs slightly better than POMCP-ER and is able to successfully pack over half of the groceries before the 15 minute timeout.

FF-REPLAN chooses the most likely hypothesis and disregards the inherent entropy in the state space. As a result, FF-REPLAN makes a mistake when the most likely hypothesis does not correspond to the true state. On the other hand by employing the belief space representation of belief space planning, LESAMPLE is able to maintain a belief of the various scene hypotheses and update this belief in the next replanning cycle even after it samples a false hypothesis. This makes LESAMPLE more robust to noisy state estimation. It is worth noting that, in scenarios where the most likely hypothesis of the scene estimate represents the true scene graph, FF-REPLAN performs less number of actions than LESAMPLE. This is because LESAMPLE does not always sample the true scene graph hypothesis and could potentially perform more actions than necessary. Such scenarios however do not occur often enough in the low entropy tasks to make FF-REPLAN a more efficient approach than LESAMPLE.

BPSTREAM* aggressively performs online planning. It only executes the first action in the generated plan and replans even when no mistake is committed. As such, even thoug h BPSTREAM* employs the belief space representation and samples from the belief space, it ends up planning much more often LESAMPLE, resulting

Algorithm	Total time spent(s)	Avg. time per action	Avg. num. packed items
LESAMPLE	**411.3**	**11.5**	**8.0**
FF-Replan	658.3	13.2	8.0
BPSTREAM*	770.4	20.3	8.0
DESPOT	900.0**	71.0	4.5
POMCP-ER	900.0**	539.9	1.0

Table 3.1: Summary of results from experiments for low entropy tasks (H values between 0.3 and 0.5). **DESPOT and POMCP-ER could not complete the tasks before the 900 second timeout so we set their total time spent to 900 seconds

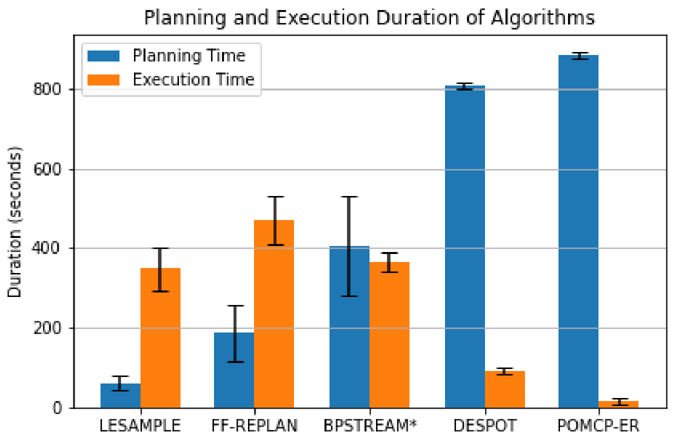


Figure 3.4: **Planning and Execution time** results for LESAMPLE and the benchmarked algorithms from performing the low entropy Grocery Packing Tasks (H values between 0.3 and 0.5). Error bars represent one standard deviation from the mean. The maximum time allocated for each task is 900 seconds.

in its higher planning time. Because it samples from the belief space, BPSTREAM* commits less mistakes than FF-REPLAN.

POMCP-ER performs the worst in planning time, execution time and number of items packed. It is unable to complete any of the packing tasks. It also packs significantly fewer objects than LESAMPLE, FF-REPLAN and BPSTREAM*. This is because of the high branching factor in the Monte Carlo search tree. DESPOT searches over a sparser tree than POMCP-ER, making it faster and better performing than POMCP-ER. DESPOT however still falls short when compared with BPSTREAM*, FF-REPLAN and LESAMPLE which perform symbolic planning on a determinized belief space.

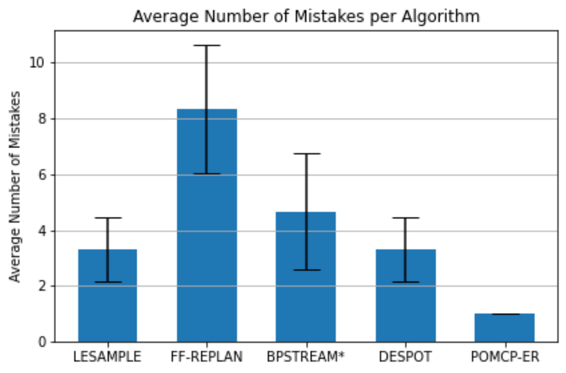


Figure 3.5: Experimental results for **Number of Mistakes** for LESAMPLE and the benchmarked algorithms from performing the low entropy Grocery Packing Tasks (H values between 0.3 and 0.5). Error bars represent one standard deviation from the mean. Note that POMCP-ER and DESPOT make few mistakes because the planning time takes up most of the allocated time per task. Hence they barely take any actions before timeout is reached.

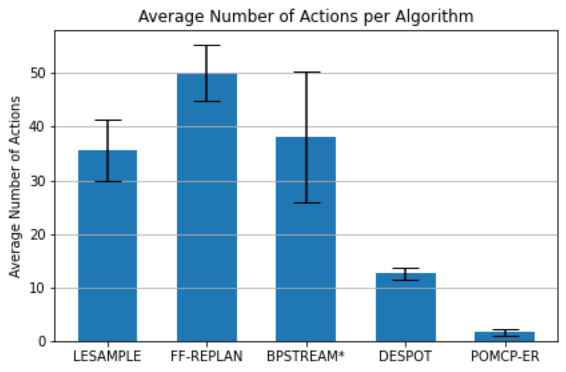


Figure 3.6: Experimental results for **Number of Actions** for LESAMPLE and the benchmarked algorithms from performing the low entropy Grocery Packing Tasks (H values between 0.3 and 0.5).

Belief Space Planning approaches like POMCP-ER and DESPOT are developed to solve problems with high entropy state spaces. As such, they spend a lot of computation in deciding the approximately optimal action to take next. This makes them inefficient for state spaces with low entropy. By adopting the fast, goal-directed features of classical planning through the use of a symbolic planner, LESAMPLE is able to efficiently solve low entropy problems by directly acting upon a sampled scene graph. This makes LESAMPLE a favorable choice for solving low perceptual entropy tasks.

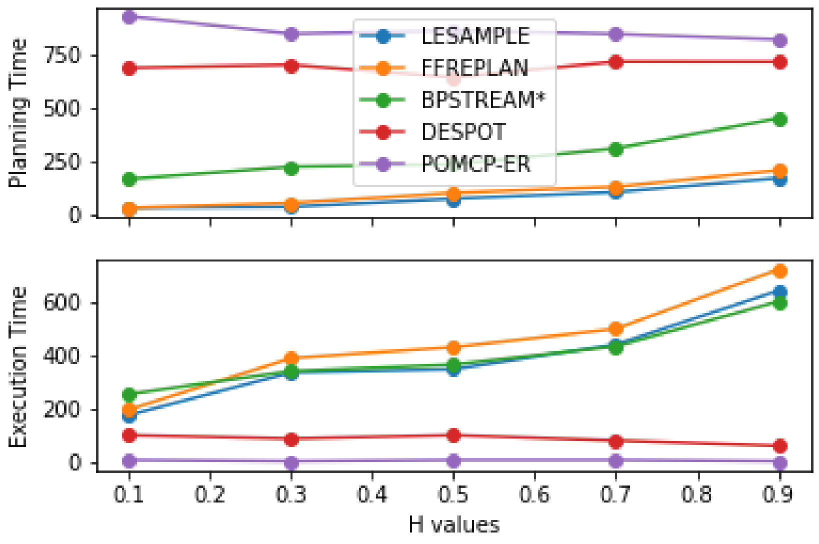


Figure 3.7: Experimental results for the **Planning Times and Execution Times** for LESAMPLE and the benchmarked algorithms from performing the Grocery Packing Task for increasing entropy values (H values from 0.1 to 0.9). The maximum time allocated for each task is 900 seconds. DESPOT and POMCP-ER do not complete any of the tasks before the 900 second time-out for any of the H values.

We also compare the average planning and execution times of LESAMPLE with the benchmarked algorithms for tasks with increasing perceptual entropy as shown in the results in Figure 3.7. FF-REPLAN has the least planning time at the lowest H value because at such entropy levels, FF-REPLAN makes little to no mistakes and as such, does not have to re-plan often. As the H value increases however FF-REPLAN expends more planning and execution time than LESAMPLE because it makes more mistakes and replans more often. BPSTREAM* consistently spends more planning time than LESAMPLE across all H values because it, by design, replans after every action, whether or not a mistake is committed. DESPOT and POMCP-ER do not complete any of the tasks across the H values before the 900 second time-out, resulting in their relatively lower execution times. Even though DESPOT and POMCP-ER are belief space planning approaches designed for high entropy state spaces, they still spend significant amounts of time to plan for a single action and are, as a result, out-performed by fast replanning approaches even in high entropy scenarios in our experiments. As such, for reversible tasks like grocery packing where action effects could be reversed through taking other actions, it is more efficient in the long run to employ fast replanning approaches that are able to quickly plan actions and replan to recover from mistakes. This resonates with results by Little et. al. [69]. However, for tasks where action effects are irreversible or where the penalty for making mistakes is significant, the more deliberative belief space planning approaches like DESPOT and POMCP-ER are more appropriate.

3.8 Conclusion

We presented LESAMPLE as an online planning method for efficiently solving sequential decision-making tasks with low perceptual entropy. The key idea is to use classical planning on estimates resulting from belief space inference over perceptual observations. As a result, LESAMPLE can perform more efficient goal-directed reasoning under scenarios of low-entropy perception. We demonstrated the efficiency of this method on grocery packing tasks. LESAMPLE demonstrated advantages in low-entropy scenarios where classical planning cannot handle uncertainty and belief space planning is unnecessarily computationally expensive.

CHAPTER IV

Probabilistic Inference in Planning for Partially Observable Long-Horizon Problems

For autonomous service robots to successfully perform long horizon tasks in the real world, they must act intelligently in partially observable environments. Most Task and Motion Planning approaches assume full observability of their state space, making them ineffective in stochastic and partially observable domains that reflect the uncertainties in the real world. We propose an online planning and execution approach for performing long horizon tasks in partially observable domains. Given the robot's belief and a plan skeleton composed of symbolic actions, our approach grounds each symbolic action by inferring continuous action parameters needed to execute the plan successfully. To achieve this, we formulate the problem of joint inference of action parameters as a Hybrid Constraint Satisfaction Problem (H-CSP) and solve the H-CSP using Belief Propagation. The robot executes the resulting parameterized actions, updates its belief of the world and replans when necessary. Our approach is able to efficiently solve partially observable tasks in a realistic kitchen simulation environment. Our approach outperformed an adaptation of the state-of-the-art method across our experiments.

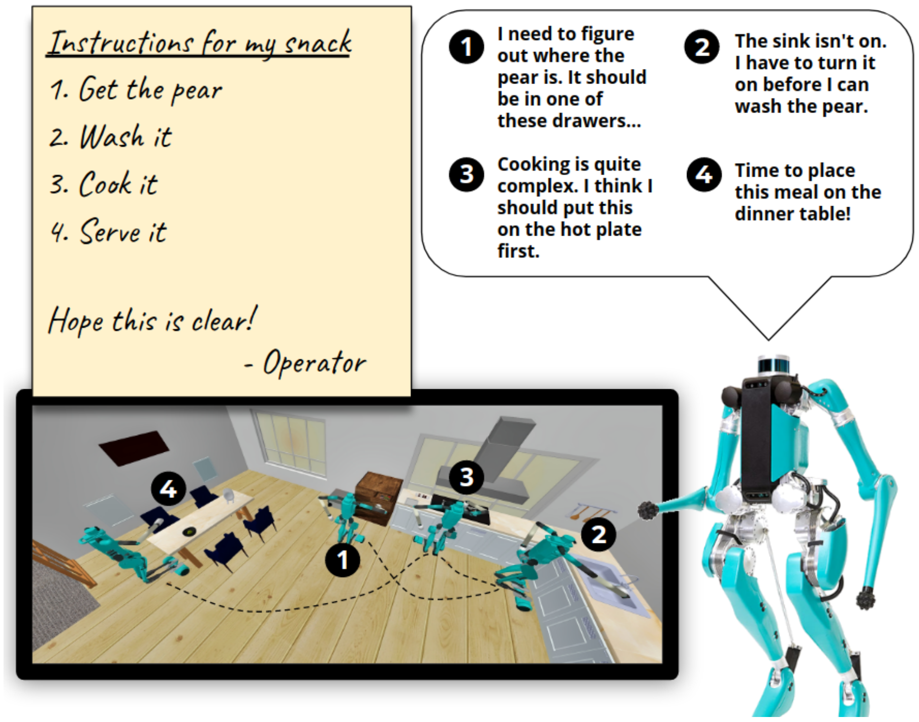

Instructions for my snack
1. Get the pear
2. Wash it
3. Cook it
4. Serve it
Hope this is clear!
- Operator
1 I need to figure out where the pear is. It should be in one of these drawers...
2 The sink isn't on. I have to turn it on before I can wash the pear.
3 Cooking is quite complex. I think I should put this on the hot plate first.
4 Time to place this meal on the dinner table!
1
2
3
4

Figure 4.1: Given vague goals to make a snack, the robot generates and executes a coherent plan to successfully complete the assigned task.

4.1 Introduction

Autonomous service robots have the potential to perform long horizon tasks such as cooking meals in restaurants and homes and setting tables. In order for this potential to be realized, such robots would have to plan actions over large state and time horizons. They would also have to account for the uncertainties in their perception and knowledge of their environment. To ensure tractability, planning for such long horizon tasks is often decomposed into planning for symbolic actions and for continuous motions. The class of approaches that interleave symbolic and continuous planning is called integrated Task and Motion Planning (TAMP) [57, 34, 88, 35, 96, 32].

Major challenges that robots planning and acting in the real world face are the uncertainty in the robot's knowledge of the current state of the world and uncertainty in the effects of the robot's actions on the future state of the world. If these uncertainties are not accounted for when planning, the robot is likely to fail to accomplish the task at hand. Most Task and Motion Planning approaches [57, 34, 88, 35, 96, 32] assume full observability of their state space leading them to fail in stochastic and partially observable domains that reflect the uncertainties in the real world.

We propose *Satisfying HYbrid COnstraints with Belief pRopAgation* (SHY-COBRA) as an approach for planning for long horizon partially observable TAMP problems. SHY-COBRA takes as inputs the robot's noisy belief of the state of the world and a plan skeleton composed of symbolic actions that achieve a specified goal. SHY-COBRA then infers satisfying parameter values for the actions needed to execute the plan successfully in partially observable domains.

Given the robot's noisy belief of the state of the world and a plan skeleton composed of symbolic actions, SHY-COBRA formulates the problem of joint inference of action parameters as a Hybrid Constraint Satisfaction problem (H-CSP)[70], which is represented as a factor graph. The factors in the factor graph are the action constraints whilst the variables are the symbolic and continuous action parameters to be inferred. The continuous parameters are initialized by the robot's noisy belief of its environment. In most TAMP approaches [57, 34, 88, 35, 96, 32], the H-CSP is solved either by sampling or by constrained optimization. Neither of these approaches explicitly accounts for the uncertainty distributions of the continuous action parameters such as uncertainty in the pose estimates and in the robot's joint configurations. SHY-COBRA instead solves the H-CSP using Pull Message Passing for Nonparametric Belief Propagation [25, 24] because of its natural ability to account for the arbitrary uncertainty models of the continuous variables. The robot executes the actions in turn and replans when necessary.

We demonstrate our approach on several simulated partially observable long horizon tasks in a realistic simulation environment as shown in Figure 4.1.

4.2 Related Work

Our work focuses on the problem of planning for long horizon tasks. The class of approaches that interleave symbolic and continuous planning is called Task and Motion Planning (TAMP). Fully observable TAMP algorithms [34, 57, 64, 70, 96, 35] assume that the agent has full knowledge of its deterministic domain and that the agent's actions have deterministic outcomes on its environment. These assumptions are however not representative of the kinds of domains robots operate in the real world, which are often stochastic and partially observable. In such domains, robots require the ability to plan in the face of incomplete knowledge and stochasticity in the effects of their actions. Relatively few methods in TAMP literature have attempted to solve these types of TAMP problems [59], [42], [78], [37].

Partially observable TAMP problems are often formulated as Hybrid Constraint Satisfaction Problems[70] and solved either through sampling [37, 42] or constraint-optimization methods [78]. Such approaches will often attempt to determinize the belief via the Maximum Likelihood Observation [42, 90] rather than incorporating the entire distribution which provides richer information. Garrett et. al.(SS-Replan) [37] represent and update the belief over object poses using particle filtering. However, their approach limits the scope of partial observability to that of object poses. In contrast, through the use of Nonparametric Belief Propagation [25, 24] on the constraint network, our approach provides the avenue for incorporating arbitrary uncertainty models on any of the variables whose value is to be inferred. We evaluate our approach against SS-Replan [37] in our experiments.

A number of prior works [73, 72] have successfully solved Constraint Satisfaction Problems using Belief Propagation. Moon et. al. [73] formulates Sudoku Solving as a Constraint Satisfaction Problem and encodes it as a factor graph. The factors in this factor graph are row, column and 3x3 cell constraints and the variables are individual cells. They successfully infer satisfying cell values for empty cells in the

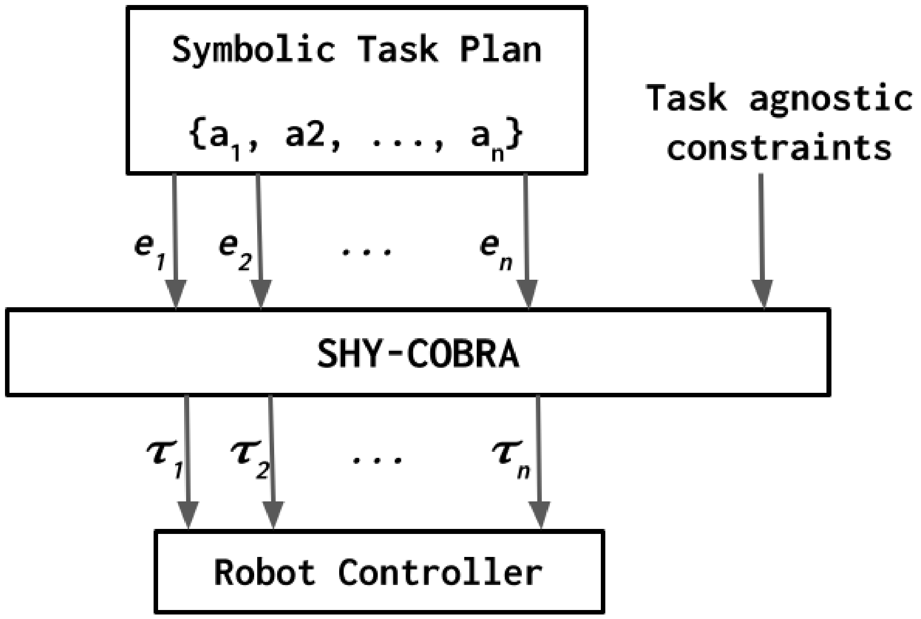


Figure 4.2: Outline of our approach. $\{e_1, e_2, \ldots, e_n\}$ are the symbolic effects of actions $\{a_1, a_2, \ldots, a_n\}$. $\{\tau_1, \tau_2, \ldots, \tau_n\}$ are inferred trajectories sent to the robot controller for execution.

sudoku puzzle.

There exists much work in the domain of structuring planning problems as ones of inference [95, 10, 8]. Toussaint et. al. [95] uses structured Dynamic Bayesian Networks to represent structured planning domain and employs loopy belief propagation to solve short-horizon reaching tasks under collision avoidance constraints with a humanoid upper body. Our approach formulates Task and Motion Planning as a Hybrid Constraint Satisfaction Problem (H-CSP) and uses Pull Message Passing Nonparametric Belief Propagation (PMPNBP) [25] to infer maximum joint beliefs that solve the H-CSP.

4.3 Problem Formulation

Given an initial belief $\mathcal{B}_0 = \{\mathcal{B}_0^O, \mathcal{B}_0^\phi\}$ of object poses and robot joint angles, and a symbolic plan skeleton $\{a_1, \cdots, a_n\}$, we aim to jointly ground each symbolic action a_k into a robot pose ϕ_k in configuration space, along with the robot trajectory $\tau_{k,k+1}$

that takes the robot from pose ϕ_k to ϕ_{k+1}. The robot can then sequentially execute the generated $\tau_{k,k+1}$ to achieve the end effect of each symbolic action a_k. After each trajectory execution, the robot perceives and updates the belief $\mathcal{B}_k$, and replans if the updated belief does not satisfy the desired end effect of the corresponding symbolic action.

We formulate the problem of jointly grounding a symbolic plan skeleton into a sequence of target robot poses $\{\phi_1, \cdots, \phi_n\}$ along with the in-between trajectories $\{\tau_{k,k+1} | k = [1, n-1]\}$ as a Hybrid Constraint Satisfaction Problem [70], where the constraints are imposed by the desired end effects of each symbolic action in the given plan, as well as other task-agnostic constraints such as collision-free and motion cost constraints.

Algorithm 2: SHY-COBRA

```
   Input: High-level Goals, $G$, Initial Belief, $\mathcal{B}_0$
1  $\Pi \leftarrow$ SymbolicPlanner ($G$) Generate a plan skeleton $\Pi$ that achieves goal
     $G$ using a symbolic planner
2  Function SHY-COBRA($\Pi$, $\mathcal{B}_0$):
3      $G \leftarrow$ ConstraintNetwork($\Pi$) Convert $\Pi$ into Constraint Network $G$
4      $G_{init} \leftarrow$ Initialize ($G, \mathcal{B}_0$) Initialize variable nodes in $G$ with $\mathcal{B}_0$
5      $G_{conv} \leftarrow$ PMPNBP ($G_{init}$) Pass messages across $G_{init}$ using PMPNBP
         algorithm until convergence
6      $\Pi_s \leftarrow$ MaxSamples ($G_{conv}, \Pi$) Set variable params in $\Pi$ with the
         max-product assignment from corresponding nodes in $G_{conv}$
7      foreach $a \in \Pi_s$ do
8          $o \leftarrow$ ExecuteAction($a$) Receive observation $o$
9          $\mathcal{B}_{current} \leftarrow$ UpdateBelief($a, o$)
10         if $o \neq$ ExpectedEffect($a$) then
11             $\Pi \leftarrow$ UpdatePlanSkel. ($\Pi_s, a$) Update plan skeleton to reflect
                 current state after executing a
12             SHY-COBRA($\Pi, \mathcal{B}_{current}$)
13             return
14         end
15     end
16     return
17 End Function
```

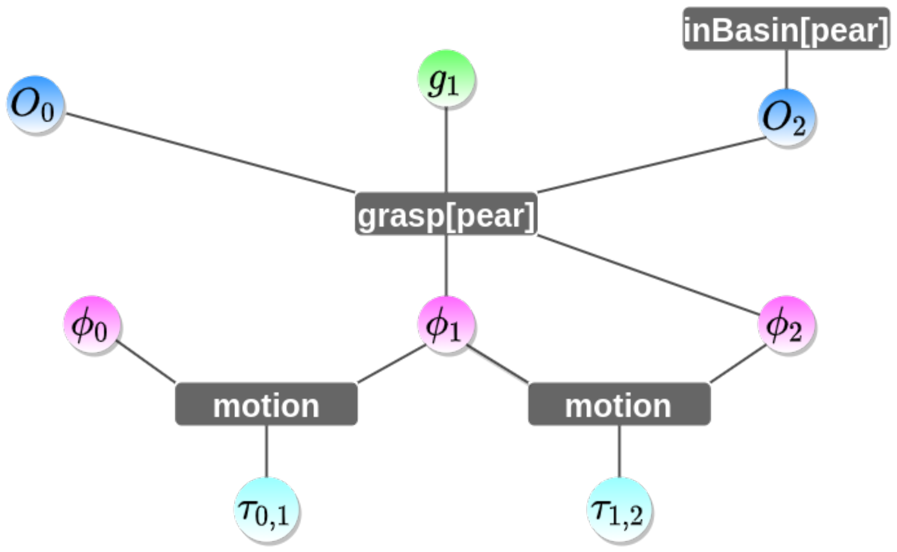


Figure 4.3: A Constraint Network. Given a task plan $\{a_1, a_2\}$ composed of actions to pick up the pear (a_1), and to transport the pear to the basin of a sink (a_2), we form a factor graph that imposes constraints (*rectangular nodes*) as factors on the variable nodes (*circular nodes*) that represent object poses, robot configurations and grasp poses. In this example, ϕ_0, ϕ_1, ϕ_2 represent the initial robot configuration and target robot configuration for action a_1, a_2, respectively. o_0, o_2 represents the initial pear pose and target pear pose after the execution of action a_2. g_1 represents a grasp pose to grasp the pear. Note that o_1 is not included in this factor graph because it has the same value as o_0. Each target robot pose is also connected with factors that express the kinematic feasibility and collision-free constraints. We do not show these factors for clarity. The grasp stability constraint on g_1 is also not shown in this diagram. The motion constraint node connected to variable nodes ϕ_0, ϕ_1 and $\tau_{(0,1)}$ encourages $\tau_{(0,1)}$ to be the shortest trajectory between configurations ϕ_0 and ϕ_1. PMPNBP is run on this factor graph to jointly infer satisfying action parameters.

4.4 Methodology

4.4.1 Satisfying Hybrid Constraints with Belief Propagation (SHY-COBRA)

As described in Algorithm 2 and Figure 4.2, SHY-COBRA takes as input, the robot's noisy belief of the state of the world and a plan skeleton [70] composed of symbolic actions. The plan skeleton is obtained by a symbolic planner that plans actions to achieve specified goal(s). Actions in this plan skeleton have free parameters like grasp poses and arm trajectories whose values are needed to be able to execute these actions in the world. We formulate the problem of inferring the values of the free parameters as a Hybrid Constraint Satisfaction Problem (H-CSP) (Section 4.4.2) and represent it as a constraint network as shown in Figure 4.3. The variable nodes in the constraint network are initialized with uniformly weighted sets of samples that represent the robot's noisy belief of the value of the corresponding variable. For instance, a variable node representing a target object's pose is initialized by a set of uniformly weighted poses that represent the uncertainty distribution of the pose as estimated by the robot's noisy perception system.

The H-CSP is solved by inferring the satisfying maximum joint belief of the variable nodes using a max-product version of Pull Message Passing Nonparametric Belief Propagation (PMPNBP)[25] (Section 4.4.3). We then assign each free parameter in the plan skeleton with the max-product assignment of the belief of its corresponding variable node and execute the plan in the world.

If there is an unexpected effect of the robot's action while executing the plan, we update the plan skeleton and the constraint network and perform message passing using PMPNBP to infer the variable assignments for the new constraint network. This process continues until the robot successfully completes the task.

The following subsections describe the components of SHY-COBRA in detail.

4.4.2 Hybrid Constraint Satisfaction Problem

Finding values for action parameters in a plan skeleton that satisfy all the sets of constraints is a *Hybrid Constraint Satisfaction Problem* (H-CSP). The joint set of action parameters and constraints of a plan skeleton form a factor graph called a constraint network [21, 64]. A constraint network is a factor graph with constraints as Factor nodes and action parameters as Variable nodes. Edges exist between constraints and their corresponding action parameters as depicted in Figure 4.3. Formally, an H-CSP is represented as

$$G(X_1, X_2, ..., X_r) \sim \prod_{j=1}^{N} f_j(S_j) \tag{4.1}$$

where $S_j \subseteq \{X_1, ..., X_r\}$, a subset of action parameter variable nodes that are subject to constraint factor node f_j, G is the factor graph and N is the number of factors.

To solve the H-CSP is to infer values for all action parameters that satisfy their corresponding constraints. In this work, we propose to solve the H-CSP by performing max-product Nonparameteric Belief Propagation on the constraint network that represents the plan skeleton. The following subsections describe the Nonparametric Belief Propagation algorithm we use, our message-passing scheme and how the values of the action parameters are inferred from noisy observations and partial knowledge of the robot's environment.

4.4.3 Solving the H-CSP using PMPNBP

To solve the H-CSP, we use PMPNBP to infer action parameter values that satisfy all the constraints in the constraint network. Mathematically, this is equivalent to inferring the action parameter assignments that maximize the joint probability

$$P(X) = \frac{1}{Z} \cdot \prod_{j=1}^{N} f_j(S_j), S_j \subseteq \{X_1, ..., X_r\} \tag{4.2}$$

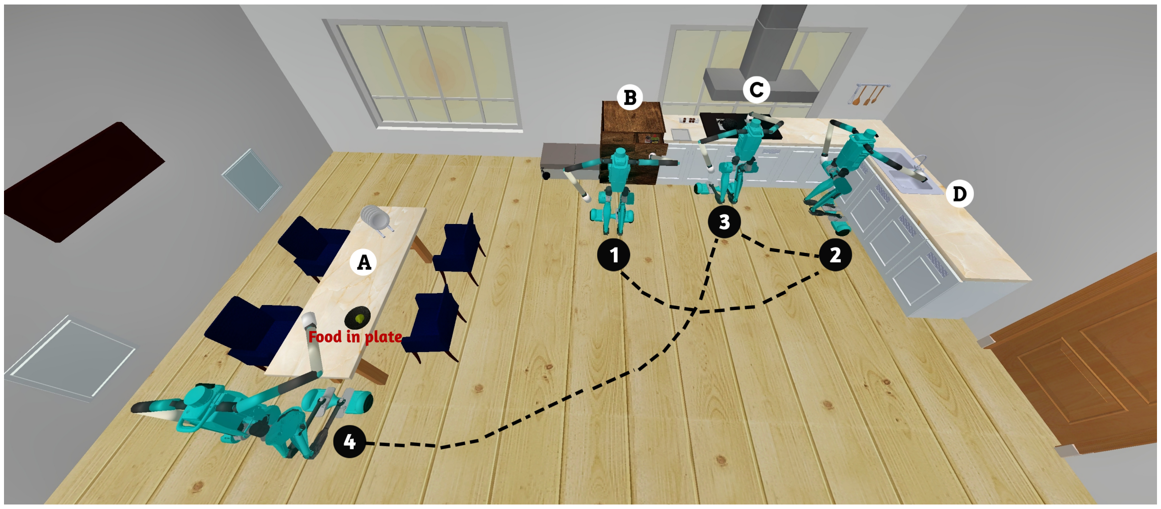


Figure 4.4: The kitchen simulation environment. The area annotated **(A)** is the dining area, **(B)** is the grocery cabinet, **(C)** is the cooking area and **(D)** is the sink area. To make a pear dinner according to an optimal plan from the SHY-COBRA planner, the robot first moves to the grocery cabinet, opens one of the drawers in the cabinet and inspects it for the pear. If the pear is not found in the drawer, the robot updates its belief of the location of the pear and replans to inspect a different drawer. If the pear is found, the robot **(1)** picks up the pear, **(2)** moves to the sink, turns on the tap and washes the pear. The robot then **(3)** moves to the stove and cook the pear. After cooking the pear, the robot **(4)** finally moves to the dining table and serves the meal.

where Z is a normalizing constant, $S_j \subseteq \{X_1, ..., X_r\}$, a subset of action parameter variable nodes that are subject to constraint factor node f_j and N is the number of factors. In the context of our work, Xs include the robot pose ϕ_ks in configuration space as well as robot trajectories $\tau_{k,k+1}$ as discussed in Section 4.3.

To infer satisfying variable assignments, we perform message passing on the constraint network with max-product PMPNBP [25]. At the beginning, the belief of each variable node is initialized with uniformly weighted samples generated by specialized generators operating on the robot's initial belief. These generators used in this work are described in detail in Section 4.5.3.

Message passing on a constraint network involves two kinds of messages; the `Constraint-to-variable` message and the `Variable-to-constraint` message.

The `Constraint-to-variable` message $msg_{f(x)\to x}$ for iteration m is computed as

$$\{\mu_x^{(i)}\}_{i=1}^M \sim bel^{m-1}(x)$$

$$\{w_x^{(i)}\}_{i=1}^M = \{\max_{\substack{y\in\rho(f)\setminus\{x\} \\ j\in 1,\cdots,M}} \sigma_f(\mu_x^{(i)}, \mu_y^{(j)})\}_{i=1}^M$$
$$msg_{f\to x}^m = \{(\mu_x^{(i)}, w_x^{(i)}) : 1 \leq i \leq M\} \tag{4.3}$$

where $y \in \rho(f) \setminus \{x\}$ represents the messages from variable nodes with edges to constraint node f except variable node x

As formulated above, M samples are drawn from the belief distribution of node x from the previous iteration, $m - 1$. The constraint function σ_f of the constraint node f is then used to compute weights for each of these samples. These weighted samples, which are now the `Constraint-to-variable` message are then passed to node x. Details on constraint functions can be found in Section 4.4.4.

The `Variable-to-constraint` message $msg_{x \rightarrow f(x)}$ for iteration m is computed as

$$msg_{x \rightarrow f}^{m} = \bigcup_{y \in \rho(x) \setminus \{f\}} msg_{y \rightarrow x}^{m-1} \tag{4.4}$$

Where $\rho(x) \setminus \{f\}$ are all the constraint nodes with edges to the variable node x except constraint node f.

As formulated above, to approximate the product of incoming messages, we take the union of all incoming messages from neighboring constraint nodes except constraint node f and normalize their weights. The resulting messages are then resampled and passed to constraint node f.

One iteration of message passing on the constraint network follows the following sequence:

1. Pass `Variable-to-constraint` messages from all variables to their corresponding constraints

2. Pass `Constraint-to-variable` messages from all constraints to their corresponding variables

After each iteration of message passing, the belief of each variable node is updated by taking the union of all incoming messages to the variable node, normalizing their weights and resampling a new set $\{\mu_x^{(i)}\}_{i=1}^{M}$ to represent the belief of the variable node.

Message passing is performed for several iterations until the maximum joint belief of each variable node converges.

After convergence, each action parameter x_i is assigned with the max-product assignment of the belief of its variable node.

4.4.4 Constraints and Constraint functions

4.4.4.1 Constraints

The constraints used in this work are

- `Motion` constraints that enforce that τ is the shortest trajectory from one robot configuration ϕ_0 to another robot configuration ϕ_1
- `Kin` constraints that enforce kinematic feasibility of the robot in configuration ϕ holding an object with grasp g
- `CfreeH` constraints that enforce that when the robot is holding an object in grasp g, the trajectory τ that the object is moved through is collision free
- `GraspH` constraints that enforce that a grasp g is a stable grasp pose
- `grasp` constraints that enforce that at a configuration ϕ_0, when a robot picks an object at position $p0$ with grasp $g0$, the grasp $g0$ will be feasible at a later time when the robot at configuration ϕ_1 places the object at position $p1$.
- `Stable` constraints that enforce that a placement pose p of an object is stable and won't cause the object to fall off.
- `inBasin` constraints that enforce that the object is placed in a stable pose in the basin
- `inSaucepan` constraints that enforce that the object is placed in a stable pose on the saucepan.

4.4.4.2 Constraint functions

A constraint function assigns weights to samples drawn from the belief distribution of the target variable node when the `Constraint-to-variable` message is computed.

The weight of each sample drawn from a variable node a is computed as follows:

$$w_a^i = \sigma_f(x_a^i, \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{T-1}) \tag{4.5}$$

where w_a^i is the weight of sample i drawn from variable node a, σ_f is the constraint function of constraint node f, T is the number of variable nodes connected by an edge to the constraint node f and $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{T-1}$ are the highest weighted samples from messages received from the $T-1$ other variable nodes connected to f.

Each type of constraint node has a unique constraint function for assessing the weight of a sample.

Constraint function example: Consider a collision-free constraint node f_{Cfree} sending a message to an arm-trajectory variable node X_{Traj}. The collision-free constraint node has edges to both an arm-trajectory variable node and a grasp pose variable node. To compute the `Constraint-to-variable` message to be sent to the arm-trajectory variable node, M samples are first drawn from the belief of the arm-trajectory variable node. To weight each arm-trajectory sample, the constraint function σ_{Cfree} takes as inputs the sample arm-trajectory x_{traj}^i and the highest weighted grasp pose sample $\hat{x}_{grasp}$. σ_{Cfree} then computes the weight of the sample arm-trajectory as

$$w_{traj}^i = C_1 \cdot \epsilon_1(x_{traj}^i) + C_2 \cdot \frac{1}{\epsilon_2(x_{traj}^i, \hat{x}_{grasp})} \tag{4.6}$$

where w_{traj}^i is the computed weight of sample x_{traj}^i, C_1 and C_2 are user-defined constants, ϵ_1 is a routine that computes the cumulative distance along x_{traj}^i from obstacles in its environment and ϵ_2 is a routine that computes the distance between the end-effector pose after travelling along x_{traj}^i and the highest weighted grasp pose $\hat{x}_{grasp}$.

The weights for all arm-trajectory samples are computed and normalized, constituting the `Constraint-to-variable` message sent to the arm-trajectory variable node.

Some other constraint functions used in this work are the

- Kinematic constraint function (σ_{kin}), which weights robot joint configuration samples and grasp poses based on how kinematically feasible a joint configuration of the robot's arm is if it is holding an object with a particular grasp pose.

- Stable constraint function (σ_{stable}), which weights placement poses based on how stable they are. i.e. how geometrically stable an object placed on a surface in a specific pose is.

- Grasp constraint function (σ_{grasp}), which weights robot configurations, initial object poses and grasp poses based on how well they are jointly feasible and how well they satisfy later target object poses.

- Motion constraint function (σ_{motion}), which weights trajectories, initial configuration and final configuration based on how short the trajectory from the initial configuration to the final configuration is.

- Grasp stability constraint function (σ_{GraspH}), which weights grasp poses based on how geometrically stable they are.

- inBasin constraint function, which weights placement poses in the basin based on how stable they are.

- inSaucepan constraint function, which weights placement poses in the saucepan based on how stable they are.

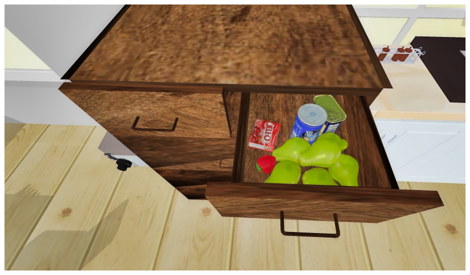
4.5 Implementation

4.5.1 Action Schemas

Actions that make up a plan are described by continuous action parameters, constraints, preconditions and effects. `preconditions` are the conditions that need to be True before an action can be executed. `effects` describe the changes in a subset of the state after an action is executed. Continuous action `parameters` are the continuous values needed by the robot to execute the action in the world. These include object poses, grasps, robot configurations and trajectories. `constraints` must hold true for all continuous parameters for the action to be executed successfully in the world. We used Fast Downward [45] as the symbolic planner for planning action schemas. The types of constraints used in the composition of action schemas in this work are described in Section 4.4.4.

Some examples of action schemas [83] used in this work are as follows:

```
 (:action pick[obj]
:parameters (φ, p, g, τ)
:constraints CFree(τ), Stable[obj](p), GraspH[obj](g) Kin[obj](φ, p, g)
:preconditions (and (at robot φ) (at obj p) (handempty))
:effects (and (holding obj) (not (handempty)))
)
 (:action place[obj]
:parameters (φ, p, g, τ)
:constraints CFree(τ), Stable[obj](p), GraspH[obj](g) Kin[obj](φ, p, g)
:preconditions (and (holding obj) (at robot φ) (at obj p))
:effects (and (not holding obj) (handempty))
)
 (:action wash[obj]
:parameters (p)
```


((a))

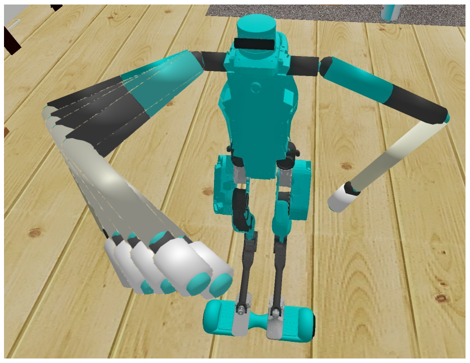

((b))

Figure 4.5: (a) Noisy pose estimate of a single pear in the cabinet drawer. (b) Noisy joint configuration estimate of the robot's right arm. Each pear in (a) and arm configuration in (b) represents a likely pose of the pear or arm joint configuration sampled from their respective estimated noise distributions.

```
:constraints Stable[obj](p), InBasin[obj](p)

:preconditions (at obj p)

:effects (clean obj)

)
```

where `obj` represents the target object and ϕ, `p`, `g`, τ represent robot joint-configuration, object pose, grasp and joint trajectory respectively.

4.5.2 Uncertainty sources and uncertainty distributions

The major advantage of SHY-COBRA over Garrett et. al. [37] is the ability of SHY-COBRA to concurrently and seamlessly incorporate arbitrary uncertainty sources and distributions. In our experiments, we consider pose estimation uncertainty and robot arm joint-configuration uncertainty as depicted in Figure 4.5. We assume that the uncertainties in pose estimation and joint configuration estimation are Gaussian distributed with zero means and variances δ_p^2 and δ_c^2 respectively. It is worth noting that SHY-COBRA is agnostic to the type of uncertainty distribution of an action parameter variable.

The robot is equipped with a perception system that updates the robot's belief after every action.

4.5.3 Computing initial samples

We compute initial samples for the various free variables by using their corresponding specialized generators. Each free variable comes with a generator that computes samples for the free variable given the initial belief.

Consider an action that picks up a target object, as described in Section 4.5.1 above. This action takes as parameters a pose variable, a grasp variable, an arm-trajectory variable and a joint configuration variable.

The initial samples for the pose variable node in the corresponding plan skeleton consist of uniformly weighted poses received from the robot's perception system's noisy estimation of the pose of the object.
Likewise, the initial grasp samples are also generated by a grasp generator which computes valid uniformly weighted grasps of the object at each of the initial sample poses. The initial arm-trajectory samples are generated by a specialized arm-trajectory generator equipped with the RRT-Connect [62] motion planner. This generator generates uniformly-weighted joint trajectories to each of the grasp samples.

Finally, to generate robot joint configuration samples, the joint-configuration generator, which is equipped with the IKFast [26] inverse kinematics solver, generates joint configuration samples to each of the grasp samples. With each of these joint configuration samples as a mean, we further sample sub joint configurations from the Gaussian distributed joint configuration noise with variance δ_c^2. During message passing, the weight w_i of each sample ϕ_i is computed as

$$w_i = \sum_j p(\phi_i^j) * \sigma_f(\phi_i^j, \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{T-1})$$

where ϕ_i^j is the joint configuration sampled from the Gaussian distribution with mean ϕ_i and variance δ_c^2, $p(\phi_i^j)$ is the Gaussian probability of ϕ_i^j, σ_f is the constraint function of the constraint node connected to ϕ_i and $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{T-1}$ are the highest weighted samples from messages received from the $T-1$ other variable nodes connected to f.

4.6 Experiments

We performed experiments on 12 randomly generated problems for 4 different tasks as described in Section 4.6.1. The experiments were performed with a simulated Digit robot [81] with the PyBullet simulation software [20] as shown in Figure 4.4. We used IKFast [26] to compute inverse kinematics solutions for the robot arms and used the pybullet_planning package [31] for motion planning.

We quantitatively compare SHY-COBRA with *SS-Replan**, a variation of the SS-Replan algorithm [37] which uses off-the-shelf RRT-Connect [62] and IKFast [26] for motion planning and inverse kinematics respectively.

For each cycle of planning, we generate 100 samples from the uncertainty distributions of object pose estimate and 100 samples from the joint configuration estimate. We run PMPNBP for 10 iterations on each cycle. The pose estimate noise is Gaussian distributed with zero mean and δ_p = 10cm standard deviation. The joint configuration estimate noise is Gaussian distributed with zero mean and δ_c = 0.25 radians standard deviation. The experiments were run on a laptop with 2.21GHz Intel Core i7 CPU, 32GB RAM and a GTX 1070 GPU.

4.6.1 Tasks

We evaluated SHY-COBRA and *SS-Replan** on 12 randomly generated problems of 4 different tasks. See the accompanying video for demonstrations of the tasks in simulation. The tasks are described as follows:

4.6.1.1 Retrieve

The high-level goal for this task is to retrieve the pear. The prior location of the pear is uniformly distributed across the 3 grocery drawers. Because of occlusion and poor lighting in the drawer, the robot has to deal with the noisy estimates of the pose of the pear as well. A successful plan opens a drawer at random and inspects it. If the pear is located, its done. Else, it updates its belief of the location of the pear and repeats the process until the pear is located. It then picks up the pear.

4.6.1.2 Wash

The high-level goal for this task is to retrieve the pear and wash it. This task has the same prior belief as the task above. A successful plan performs the *Retrieve* task as described above, sends the pear to the sink and turns on the tap to wash it.

4.6.1.3 Cook

The high-level goal for this task is to retrieve the pear, wash it and cook it. The main challenge in this task is to infer grasps and trajectories that allow the robot to pick up a cup in a specific grasp pose that makes later actions like filling the cup with water and pouring the water in the saucepan feasible during the execution of the cooking task. This task has the same prior belief as the tasks above. A successful plan performs the *Wash* task as described above, takes the pear to the stove, puts it in the saucepan, picks up a cup, fills it with water, pours the water in the saucepan and presses the cook button on the stove to cook the pear.

4.6.1.4 Serve-meal

The high-level goal for this task is to retrieve the pear, wash it, cook it and serve it. This task has the same prior belief as the tasks above. A successful plan performs the *Cook* task as described above, picks up the cooked pear from the saucepan and

Alg:	SHY-COBRA		SS-Replan*	
Task:	Planning Time	N.E	Planning Time	N.E
Retrieve	9.37 ± 5.25	1.33 ± 1.52	16.61 ± 5.68	8.67 ± 4.93
Wash	15.72 ± 6.91	2.67 ± 3.06	21.32 ± 8.53	7.00 ± 3.60
Cook	25.15 ± 5.16	3.50 ± 2.12	29.28 ± 13.61	6.33 ± 0.58
Serve-meal	37.24 ± 10.04	7.00 ± 2.83	52.75 ± 16.82	15.5 ± 0.71

Table 4.1: Results from evaluation of SHY-COBRA and SS-Replan*. The table shows the mean planning duration in seconds (Planning Time) and the number of errors (N.E) for all 12 randomly generated problems for the 4 tasks

sets it on a tray, carries the tray to the dining table and distributes the cooked pear to the plates on the table.

4.6.2 Results

Table 4.1 shows the experimental results for SHY-COBRA and *SS-Replan** on the tasks described above.

An error occurs when the robot misses its target when picking or placing an object due to noise in the object's pose or joint configuration estimate. Based on the results, SHY-COBRA was consistently more efficient than *SS-Replan** across all 4 tasks and made slightly less errors across all 4 tasks. Since *SS-Replan** is only capable of considering object pose uncertainty and doesn't account for the uncertainty in the robot arm's joint configurations, it misses its target more often and as a result makes more errors and takes longer to plan.

In spite of its ability to concurrently incorporate arbitrary uncertainty sources and distributions due to the use of PMPNBP for inference, SHY-COBRA has a time complexity that grows with the magnitude of noise in the estimation. We demonstrate this by comparing the planning times for SHY-COBRA and SS-Replan* for increasing noise in object pose estimation and joint configuration estimation as indicated in the results in Figure 4.6.

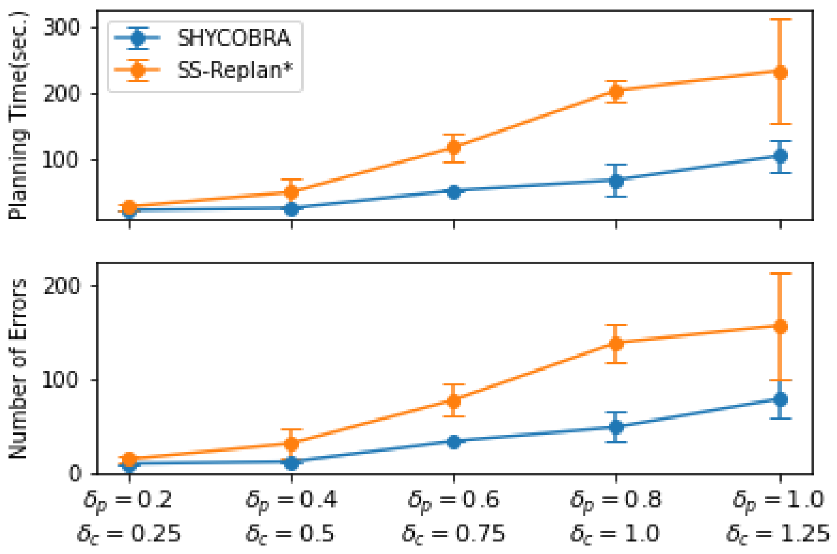


Figure 4.6: Plots comparing the planning times and number of errors of SHY-COBRA and SS-Replan* for increasing uncertainties in the pose estimation and joint configuration when performing the *Retrieve* task. δ_p represents the standard deviation of the zero mean object pose estimation noise and δ_c represents the standard deviation of the zero mean joint configuration estimation noise

4.7 Conclusion

We proposed a planning approach for long horizon planning under uncertainty. Our approach jointly infers satisfying action parameter values for a plan skeleton that are needed to successfully execute the plan in a stochastic, partially observable environment. Our approach outperformed an adaption of the SS-Replan algorithm across all tasks in our experiments.

CHAPTER V

Grounded Task Planning as Mixed Integer Programming

For robots to successfully execute tasks assigned to them, they must be capable of planning the right sequence of actions. These actions must be both optimal with respect to a specified objective and satisfy whatever constraints exist in their world. We propose an approach for robot task planning that is capable of planning the optimal sequence of grounded actions to accomplish a task given a specific objective function while satisfying all specified numerical constraints. Our approach accomplishes this by encoding the entire task planning problem as a single mixed integer convex program, which it then solves using an off-the-shelf Mixed Integer Programming solver. We evaluate our approach on several mobile manipulation tasks in both simulation and on a physical humanoid robot. Our approach is able to consistently produce optimal plans while accounting for all specified numerical constraints in the mobile manipulation tasks. Open-source implementations of the components of our approach as well as videos of robots executing planned grounded actions in both simulation and the physical world can be found at this url: https://adubredu.github.io/gtpmip

5.1 Introduction

The successful execution of manual tasks often requires the satisfaction of certain physical constraints. For instance, to retrieve a sugar canister from a seven-foot high shelf in a kitchen, the average person would have to stand *close enough* and stretch their hands *far enough* to not only reach the sugar, but to grasp it stably and lift it. Here, *close enough* and *far enough* are physical constraints that need to be satisfied to guarantee the success of their efforts to retrieve the sugar. Similarly, for a robot to successfully perform this sugar retrieval task, it would have to account for similar physical constraints when *deciding* the *right actions* to take. While deciding the right actions, this shrewd robot would also have to bias its decisions towards actions that optimize certain quantities like energy consumed or distance travelled. We call this problem the *Optimal Constrained Task Planning* problem.

The predominant way to solve a task planning problem is to formulate it as a symbolic AI planning problem, represent it in a graph structure and employ graph search algorithms to find paths from a start state to a goal state. However, this approach is purely symbolic and provides no avenues for incorporating numerical constraints in the planning process or to bias the search to choose actions that optimize numerical objective functions [7, 45, 9]. As such, these approaches only allow the specification of symbolic task goals and are unable to support the specification of continuous goals. It is often up to the human expert to introduce symbols that aptly represent desired continuous goals. A few works have proposed extensions to graph search algorithms to enable them to handle constraints and objective functions [28]. However, these approaches are only capable of handling simple additive objective functions with soft linear constraints.

In this work, we propose an approach for task planning that is capable of handling both linear and non-linear constraints and optimizes for convex objective functions to global optimality. We take a unique approach to the task planning problem by

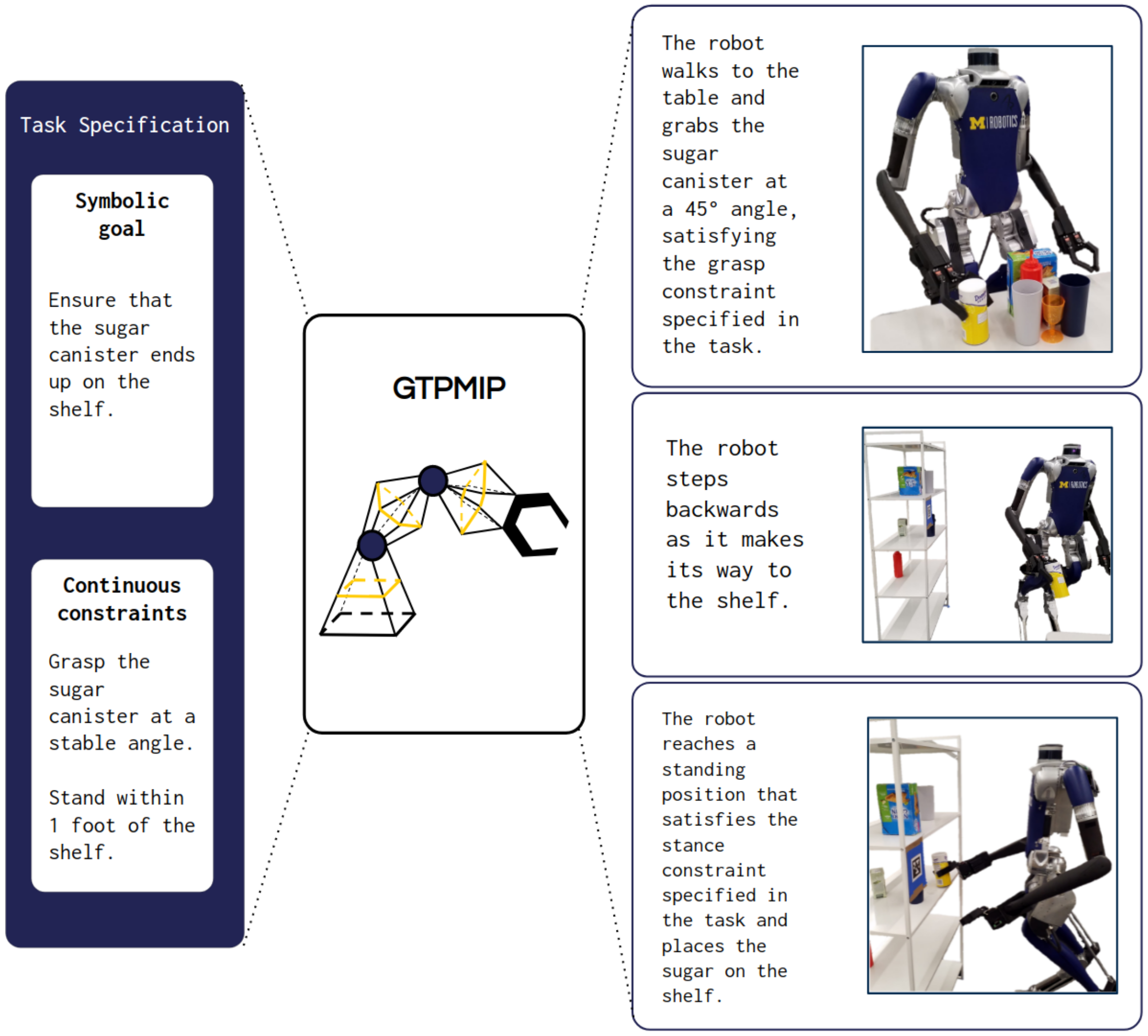

Task Specification
Symbolic goal
Ensure that the sugar canister ends up on the shelf.
Continuous constraints
Grasp the sugar canister at a stable angle.
Stand within 1 foot of the shelf.
GTPMIP
The robot walks to the table and grabs the sugar canister at a 45° angle, satisfying the grasp constraint specified in the task.
The robot steps backwards as it makes its way to the shelf.
The robot reaches a standing position that satisfies the stance constraint specified in the task and places the sugar on the shelf.

Figure 5.1: Given a constrained task planning problem, our approach (GTPMIP) plans a sequence of coherent actions with optimal parameters needed to accomplish the task.

encoding the entire task planning problem as a single Mixed Integer Convex Program (MICP). By doing this, we gain the flexibility of subjecting the problem to arbitrary action constraints that need to be satisfied in order for the resulting plan to be physically realizable by a robot. We also escape the restriction of having to specify planning goals symbolically as this encoding enables the specification of continuous planning goals. We then use an off-the-shelf Mixed Integer Programming solver to solve the MICP to optimality, extract the grounded plan and its optimal parameters and execute the plan with a robot. The unique contributions of this work are as follows:

- Firstly, we extend the Planning Domain Definition Language (PDDL) to allow for the specification of numerical action and task constraints, numerical initial values of continuous task variables, numerical objective functions, numerical action dynamics functions, numerical preconditions and numerical effects. We call this extension the Hybrid PDDL Description (HPD).

- Secondly, we propose a unique representation of continuous actions as Funnels and propose an approach for representing the continuous plan space, which we call the Hybrid Funnel Graph.

- Finally, we describe an approach for encoding the Hybrid Funnel Graph as a single Mixed Integer Convex Program which we solve using an off-the-shelf MIP solver.

We evaluate our approach in simulation on several 2-D object rearrangement task planning problems subject to unique geometric constraints. We also demonstrate our approach on real-world mobile manipulation tasks involving kinematic constraints using the Digit humanoid robot, as shown in Figure 5.1.

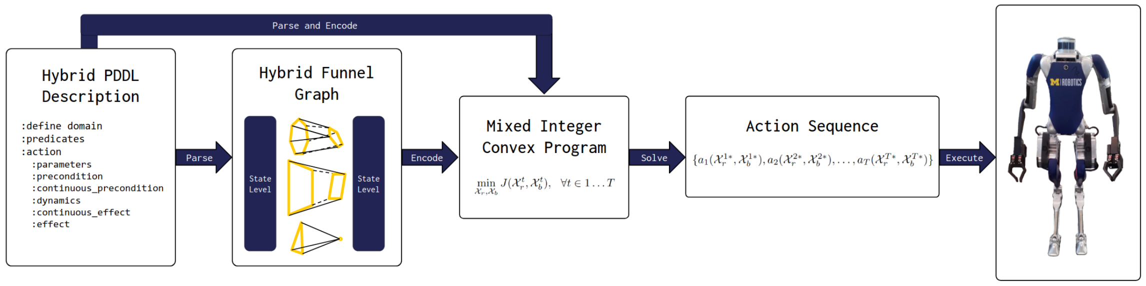

Parse and Encode
Hybrid PDDL Description
:define domain
:predicates
:action
:parameters
:precondition
:continuous_precondition
:dynamics
:continuous_effect
:effect
Parse
Hybrid Funnel Graph
State Level
State Level
Encode
Mixed Integer Convex Program
$\min_{\mathcal{X}_r,\mathcal{X}_b} J(\mathcal{X}_r^t, \mathcal{X}_b^t), \quad \forall t \in 1 \dots T$
Solve
Action Sequence
$\{a_1(\mathcal{X}_r^{1*}, \mathcal{X}_b^{1*}), a_2(\mathcal{X}_r^{2*}, \mathcal{X}_b^{2*}), \dots, a_T(\mathcal{X}_r^{T*}, \mathcal{X}_b^{T*})\}$
Execute

Figure 5.2: An overview of GTPMIP. Given a task, our approach represents the plan space as a Hybrid Funnel Graph, encodes the task and the Hybrid Funnel Graph as a Mixed Integer Convex Program and solves it to produce an optimal action sequence which is executed by the robot.

5.2 Related Works

5.2.1 Mixed Integer Programming

A Mixed Integer Program (MIP) is a mathematical optimization problem with both integer and real-valued variables [66]. The ability of MIPs to have both discrete and continuous variables makes them ideal for formulating sequential planning problems that involve taking discrete actions which are subject to continuous constraints [1, 23, 48, 97]. This work encodes the optimal constrained task planning problem as a Mixed Integer Convex Program (MICP).

5.2.2 Symbolic AI Planning

The state-of-the-art methods in STRIPS-style [29] Symbolic AI Planning first decompose the planning problem into a causal graph and employ graph search techniques like A* [43] to find plans from some initial node in the graph to a desired goal node [47, 45, 5]. Although these approaches thrive for purely symbolic domains, they do not naturally allow for the incorporation of numerical constraints and objective functions [52]. The formulation of AI Planning problems as Integer Programs has been explored a few times in the planning and scheduling literature [100, 98].

In this work, we build up on Vossen et. al's [100] Integer Programming formulation by encoding the AI Planning problem as a MIP and solving it using off-the-shelf MIP solvers. This MIP encoding is convenient because it naturally allows for the incorporation of numerical constraints and objective functions into the planning problem. This ability is essential because real world problems often involve numerical constraints and objectives.

5.2.3 Integrated Task and Motion Planning

The class of approaches that interleave symbolic AI planning and continuous planning is called Task and Motion Planning [34, 3, 35, 96, 88]. Among these, approaches like Garrett et. al. [34, 35] and Srivastava et. al. [88] devise symbols to describe continuous constraints for actions. These symbols are used as action preconditions in the symbolic AI planning process and are evaluated on demand. In addition to the chore of having to devise symbolic abstractions for every continuous constraint, these approaches are hampered by their requirement of symbolic goal descriptions. They are incapable of planning for continuous goal descriptions that can only be evaluated by an objective function. By formulating the planning problem as a Mixed Integer Convex Program, our proposed approach is able to both reason on the symbolic level using integer-valued variables and integer inequalities and optimize for continuous objective functions using the real-valued variables, and convex equations and inequalities.

5.2.4 Combined symbolic and continuous planning as Mathematical Programs

Works like Toussaint [96] and Li and Williams [67] have sought to solve the combined symbolic and continuous planning problem by formulating them as Mathematical Programs. Toussaint [96] uses an iterative 3-level Nonlinear Constrained Optimization to optimize for continuous robot configurations over discrete action sequences it acquires from running Monte Carlo Tree Search. Our approach differs from Toussaint [96] in that, we formulate the entire planning problem as a single Mixed Integer Program, solving for both the discrete action sequences and the continuous robot configurations in a single run. Li and Williams [67] employ hybrid flow graphs to represent the entire plan space and formulate the planning problem as a Mixed Logic Nonlinear Program in planning actions for autonomous underwater vehicles in ocean exploration tasks. Our representation of continuous actions as funnels and our formulation of Hybrid Funnel Graphs to represent the plan space are inspired by Li and Williams's work.

5.3 Problem Formulation

In this work, we tackle the problem of optimal task planning under numerical constraints. Our goal is to generate a grounded plan made up of a logically consistent sequence of actions, with each action associated with its corresponding optimal continuous parameters. We will use the task of package rearrangement within a warehouse environment (The Warehouseman's Problem [84]) by a mobile manipulator robot as a running example for the remainder of this paper.

The inputs to our approach are:

- A set of initial symbolic propositions, $\mathcal{I}$, that describe the initial symbolic state of the world. For example, the set

  $$\mathcal{I} = \big\{\texttt{(hand-empty)},\texttt{(not (packed boxA))}\big\}$$

  represents a world where the robot is not holding any package and `boxA` has not been packed.

- A set of initial continuous variable values, $\mathcal{X}_r^I$ and $\mathcal{X}_b^I$, where $\mathcal{X}_r^I$ represents the robot's initial configuration in SE(2) space and $\mathcal{X}_b^I$ represents the configurations of the packages, also in SE(2) space.

- A set of action primitives, $\mathcal{A}$, that can be executed by a robot. An action primitive is comprised of:
  - Symbolic preconditions: A conjunction of symbols whose truth-value must be true in order for the action to be executable.
  - Continuous preconditions: Continuous constraints on the continuous variables ($\mathcal{X}_r$ and $\mathcal{X}_b$) that must be satisfied in order for the action to be executable.
  - Action Dynamics: A dynamics function that computes the state of the continuous variables ($\mathcal{X}_r$ and $\mathcal{X}_b$) after the action is executed.
  - Symbolic effects: A conjunction of symbols that represent the state of the world after the action is executed.
  - Continuous effects: The numerical values of continuous variables ($\mathcal{X}_r$ and $\mathcal{X}_b$) after the action is executed.
- A set of task-specific numerical constraints $\mathcal{H}$ on the continuous variables.
- A set of goal symbolic propositions, $\mathcal{G}$, that must hold true at the end of the plan execution. For example, the symbolic propositions

$$\mathcal{G} = \big\{\texttt{(hand-empty)},\texttt{(packed boxA)}\big\}$$

  would represent a world where the robot is not holding any package and `boxA` is packed.
- A set of goal continuous variable values, $\mathcal{X}_r^G$ and $\mathcal{X}_b^G$.
- An objective function $J(\mathcal{X}_r, \mathcal{X}_b)$ to be optimized.

The output of our approach is a grounded plan π^* made up of a sequence of logically consistent actions $\{a_1(\mathcal{X}_r^{1*}, \mathcal{X}_b^{1*}), a_2(\mathcal{X}_r^{2*}, \mathcal{X}_b^{2*}), \ldots, a_N(\mathcal{X}_r^{N*}, \mathcal{X}_b^{N*})\}$, with each action associated with its corresponding optimal continuous parameter values.

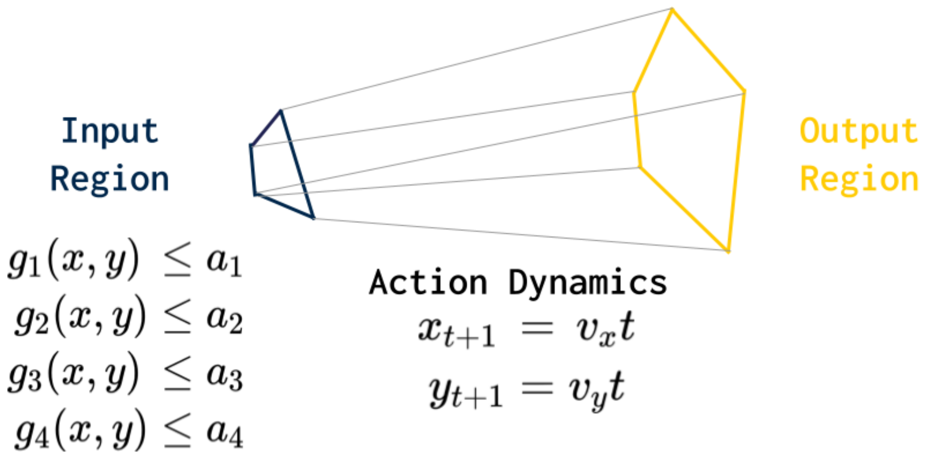


Figure 5.3: A funnel representation for the `move` action. The input region is formed by the intersection of continuous precondition inequality constraints (g_1 to g_4) on the robot position. The action dynamics compute the next robot position, x_{t+1}, y_{t+1} after the action is applied to the poses in the input region. The output region represents the space of resulting poses.

5.4 Methodology

In this section, we describe each component of our approach, as illustrated in Figure 5.2. We name our approach *Grounded Task Planning as Mixed Integer Programming* (GTPMIP). As stated in the previous section, GTPMIP takes as input a description of the optimal constrained task planning problem including descriptions of action primitives the robot is capable of executing. GTPMIP then builds a Hybrid Funnel Graph from this description to represent the entire plan space. Finally, it encodes the Hybrid Funnel Graph and the planning problem as an MICP, which it then solves using an off-the-shelf MIP solver. Each of these components are described in the following subsections.

5.4.1 Hybrid PDDL Description

Hybrid PDDL Description (HPD) is an extension of PDDL that allows for the specification of task planning problems with numerical action and task constraints, numerical initial values of continuous task variables, numerical objective functions, numerical action dynamics functions, numerical preconditions and numerical effects. Similar to PDDL, an HPD description of a task planning problem is made up of two files; the `domain.hpd` file and the `problem.hpd` file.

The `domain.hpd` file describes the action primitives that the robot can execute. An action primitive has fields

- `:action` to specify the name of the action primitive
- `:parameters` to specify the symbolic and continuous parameters the action takes.
- `:precondition` to specify a conjunction of symbols whose truth-values must be true in order for the action to be executable.
- `:continuous_precondition` to specify continuous constraints on the continuous variables that must be satisfied in order for the action to be executable.
- `:dynamics` to specify dynamics functions that compute the state of the continuous variables after the action is executed.
- `:continuous_effect` to specify the numerical values of continuous variables after the action is executed.
- `:effect` to specify a conjunction of symbols that represent the state of the world after the action is executed.

The `problem.hpd` file describes the initial symbolic and continuous states as well as the goal symbolic and continuous states of the task. It also describes the task-specific constraints and the objective function to be optimized.

5.4.2 Funnels and Hybrid Funnel Graphs

We represent action primitives as *funnels*. A funnel is made up of three components; an input region, a dynamics function and an output region. The input region is the region of intersection of all the continuous constraints that need to be satisfied before the action can be executed (the continuous preconditions). The dynamics function computes the state of the continuous variables after the action is executed. We apply the dynamics function on the peripheries of the input region to result in a new region which we call the output region. The geometric representation of this abstraction takes the shape of a funnel as shown in Figure 5.3; hence its name. The representation of action primitives as funnels helps in determining which action primitives are *applicable* given the state of the robot. If the values of the continuous variables of the current state intersects with the input region of a funnel and the symbolic preconditions of the corresponding action hold true for the symbolic propositions of current state, then the action is *applicable*. The output region of the funnel also determines the continuous state after the corresponding action is executed. In addition to action funnels, *No-op* funnels are identity operations which represent actions that make no changes to the symbolic state of the world and whose set of symbolic preconditions are equal to their set of symbolic effects.

Given this representation of actions as funnels, we build up the Hybrid Funnel Graph by alternating between state levels and action levels. A state level is a set of all possible states (both symbolic and continuous) at a specific time instance. An action level is a set of all *applicable* funnels at a specific time instance. The first state level is a set of all the symbolic propositions $\mathcal{I}$ and continuous variables $\mathcal{X}_b^I$ where $\mathcal{X}_r^I$ that make up the initial state. The continuous variables could take the form of either singular continuous values or intervals of continuous values that represent regions in the continuous space (SE(2) space in our package rearrangement problem formulation). We then compute all funnels that are applicable given the symbolic and continuous state variables in the first state level. These funnels constitute the first action level. As noted in the previous paragraph, a funnel is applicable to a state level if the continuous state variables in the state level intersect with the input region of the funnel and the symbolic preconditions of the action corresponding to the funnel hold true for the symbolic propositions of the state level. We also include to the first action level, *No-op* funnels for each symbolic proposition in the state level. The second state level is then computed as the set of all symbolic effects of actions and output regions of their corresponding funnels in the first action level. These output regions are computed by applying the funnel's dynamics function to the region of intersection of the continuous variables of the first state level and the funnel's input region. Likewise, the second action level is computed in the same manner as the first action level.

After the computation of each state level, we check if the goal symbolic propositions $\mathcal{G}$ hold true in the state level and if the goal continuous variable values $\mathcal{X}_b^G$ where $\mathcal{X}_r^G$ intersect the continuous variable regions in the state level. If both of these conditions are true, we have a valid Hybrid Funnel Graph for the task and terminate the graph building process. If not, we keep building the graph by adding additional state and action levels. Each action level represents a single time step in the resulting plan. Hence the total number of action levels represents the total period of the entire resulting plan. Note that, since the Hybrid Funnel Graph starts with the initial state level and ends with the terminal state level, the total number of state levels is greater than the total number of action levels by 1. This process is similar to the process of building planning graphs in GraphPlan [9] except that planning graphs in GraphPlan are made up of only symbolic propositions and symbolic actions. Hybrid Funnel Graphs are made up of both symbolic propositions and continuous variables, hence its name.

Unlike with GraphPlan where the graph building process is guaranteed to terminate if the planning problem is valid, our approach to building Hybrid Funnel Graphs is not guaranteed to terminate due to our inclusion of continuous variables. However in this work we observe GTPMIP successfully terminate graph building in every planning problem it is applied to.

5.4.3 Encoding as a Mixed Integer Convex Program

Before we encode the Hybrid Funnel Graph and the planning problem as an MICP, we first define a set of useful variables.

- Let T represent the total number of action levels in our Hybrid Funnel Graph, which is also the total planning period.
- Let $\mathcal{F}$ represent the set of all instantiated symbolic propositions in our planning domain. Hence $\mathcal{I} \subseteq \mathcal{F}$ and $\mathcal{G} \subseteq \mathcal{F}$
- Let $\mathcal{A}$ represent the set of all instantiated actions in our planning domain.
- Let pre_f represent the set of all actions that have symbolic proposition f as a symbolic precondition.
- Let add_f represent the set of all actions whose symbolic effects affirm symbolic proposition f. (the truth-value of f in the action's symbolic effect is **True**).
- Let del_f represent the set of all actions whose symbolic effects negate symbolic proposition f. (the truth-value of f in the action's symbolic effect is **False**)

Next, we define integer variables. For all $f \in \mathcal{F}$ and $t \in 1 \dots T$,

$$p_{f,t} = \begin{cases} 1, & \text{if proposition } f \text{ holds true at time } t \\ 0, & \text{otherwise} \end{cases}$$

$$q_{a,t} = \begin{cases} 1, & \text{if action } a \text{ is taken at time } t \\ 0, & \text{otherwise} \end{cases}$$

Given these variable definitions, we now build the constraints into our MICP.

The first set of constraints to be added are the initial and terminal constraints.

The initial constraint,

$$p_{f,1} = \begin{cases} 1, & \forall f \in \mathcal{I} \\ 0, & \forall f \notin \mathcal{I} \end{cases} \tag{5.1}$$

ensures that all initial symbolic propositions hold true in the first state level.

The terminal constraint,

$$p_{f,T+1} = 1,\ \forall f \in \mathcal{G} \tag{5.2}$$

ensures that all goal symbolic propositions hold true in the last state level.

The next set of constraints are the precondition constraints

$$q_{a,t} \leq p_{f,t},\ \forall a \in \text{pre}_f, \forall t \in 1 \ldots T, f \in \mathcal{F} \tag{5.3}$$

These inequality constraints encode the implication constraint that if action a, which has symbolic proposition f as its precondition, is taken in action level t, then f should also hold true in state level t. This constraint is called the precondition constraint because it ensures that all preconditions of an action hold true before the action can be taken.

The next set of constraints are the effect constraints

$$p_{f,t+1} \leq \sum_{a \in \text{add}_f} q_{a,t},\ \forall t \in 1 \ldots T, f \in \mathcal{F} \tag{5.4}$$

These inequality constraints encode the implication constraint that if symbolic proposition f holds true in state level $t+1$, then at least one action a which has f as a positive effect should be taken in action level t.

The next set of constraints are the mutual exclusion constraints

$$q_{a,t} + q_{a',t} \leq 1 \tag{5.5}$$

for all $t \in 1 \ldots T$ and all a, a' for which there exists an $f \in \mathcal{F}$ such that $a \in \text{del}_f$ and $a' \in \text{pre}_f \cup \text{add}_f$.
These inequality constraints ensure that two actions a and a' that cancel each other are not both taken in action level t.

The next set of constraints are the task-specific numerical constraints

$$q_{a,t} \leq h(\mathcal{X}_r^t, \mathcal{X}_b^t), \ \forall h \in \mathcal{H}, t \in 1 \ldots T \tag{5.6}$$

that ensure that if action a is taken in action level t, the continuous variable parameters of a satisfy all the task-specific numerical constraints $\mathcal{H}$.

The final set of constraints are the initial and terminal constraints

$$\mathcal{X}_r^1 = \mathcal{X}_r^I, \ \mathcal{X}_b^1 = \mathcal{X}_b^I \tag{5.7}$$

and

$$\mathcal{X}_r^{T+1} = \mathcal{X}_r^G, \ \mathcal{X}_b^{T+1} = \mathcal{X}_b^G \tag{5.8}$$

that ensure that the values of continuous variables at the first and final levels are equal to the problem-specified initial and goal continuous variable values respectively.

The objective function to be optimized

$$J(\mathcal{X}_r^t, \mathcal{X}_b^t) \ \forall t \in 1 \ldots T \tag{5.9}$$

is a convex function on all continuous variables for the entire planning period. For a warehouseman's problem, a suitable objective function would be to minimize the total Euclidean distance the robot travels while rearranging the packages.

Putting together Equations 1 - 9, our entire MICP can now be summarized as

$$
\begin{aligned}
\min_{\mathcal{X}_r,\mathcal{X}_b} J(\mathcal{X}_r^t, \mathcal{X}_b^t),\ \forall t \in 1 \dots T \\
\text{subject to} \\
p_{f,1} = \begin{cases} 1, & \forall f \in \mathcal{I} \\ 0, & \forall f \notin \mathcal{I} \end{cases} \\
p_{f,T+1} = 1,\ \forall f \in \mathcal{G} \\
q_{a,t} \le p_{f,t},\ \forall a \in \text{pre}_f,\ \forall t \in 1 \dots T, f \in \mathcal{F} \\
p_{f,t+1} \le \sum_{a \in \text{add}_f} q_{a,t},\ \forall t \in 1 \dots T, f \in \mathcal{F} \\
q_{a,t} + q_{a',t} \le 1 \\
q_{a,t} \le h(\mathcal{X}_r^t, \mathcal{X}_b^t),\ \forall h \in \mathcal{H}, t \in 1 \dots T \\
\mathcal{X}_r^1 = \mathcal{X}_r^I,\ \mathcal{X}_b^1 = \mathcal{X}_b^I \\
\mathcal{X}_r^{T+1} = \mathcal{X}_r^G,\ \mathcal{X}_b^{T+1} = \mathcal{X}_b^G \\
p \in \{0,1\}, q \in \{0,1\}, \mathcal{X} \in \text{SE}(2)
\end{aligned}
\tag{5.10}
$$

We solve this MICP using an off-the-shelf MIP solver which returns the grounded plan π^* made up of a sequence of logically consistent actions

$$\{a_1(\mathcal{X}_r^{1*}, \mathcal{X}_b^{1*}),\ a_2(\mathcal{X}_r^{2*},\ \mathcal{X}_b^{2*}),\ \dots,\ a_T(\mathcal{X}_r^{T*}, \mathcal{X}_b^{T*})\}$$

with each action associated with its corresponding optimal continuous parameter values.

5.5 Implementation

5.5.1 Open-source software implementations

We provide open-source software implementations for each of the components of our approach. We provide

- `HPD.jl` as a software package for reading and parsing HPD files. It also contains example HPD files for the warehouse rearrangement problem.
  url: https://github.com/adubredu/HPD.jl

- `HybridFunnelGraphs.jl` as a software package for building complete Hybrid Funnel Graphs when given HPD files of an optimal constrained Task Planning Problem. url: https://github.com/adubredu/HybridFunnelGraphs.jl

- `gtpmip.jl` as a software package for encoding Hybrid Funnel Graphs and HPD files of an optimal constrained task planning problem as an MIP and solving the MIP to output the optimal plan. url: https://github.com/adubredu/gtpmip.jl

- `westbrick.jl` as a simulator for a 2D version of the Warehouse rearrangement problem. url: https://github.com/adubredu/westbrick.jl

5.5.2 Solving the Mixed Integer Convex Program

Throughout our experiments, we use the Gurobi Optimization software [41] to solve all MICPs. We use Gurobi because it has state-of-the-art MIP solver implementations and is empirically the fastest MIP solver amongst all solvers considered.

5.6 Experiments

We evaluate the capabilities of GTPMIP on a series of experiments both in simulation and on a physical robot.

Algorithm	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Fast Downward [45]	0.102 ± 0.0003	0.104 ± 0.002	0.214 ± 0.001	0.218 ± 0.002	0.216 ± 0.008
Pyperplan [7]	0.167 ± 0.008	0.169 ± 0.012	0.169 ± 0.014	0.183 ± 0.012	0.190 ± 0.007
Forward Search [65]	0.0006 ± 0.004	0.0011 ± 0.001	0.0021 ± 0.0014	0.004 ± 0.002	0.008 ± 0.004
GTPMIP (ours)	0.039 ± 0.011	0.033 ± 0.0004	0.062 ± 0.009	0.172 ± 0.011	0.719 ± 0.014

Table 5.1: Comparison of planning times (in seconds) of GTPMIP with those of state-of-the-art symbolic planners on purely symbolic block stacking problems with increasing number of objects.

5.6.1 Pure Symbolic Task Planning evaluation

First, we compare the symbolic task planning capabilities of GTPMIP to the state-of-the-art symbolic planning approaches Fast-Downward [45], Pyperplan [7] and Forward Search with A* [65]. We compare the planning times of these approaches on purely symbolic block stacking tasks with increasing number of objects, with Problem 1 having 4 objects and Problem 5 having 9 objects. Table 5.1 shows the average planning times of each approach.

As can be seen from results in Table 5.1, GTPMIP is slightly faster than Fast Downward and Pyperplan on the smaller Problems 1 - 4. GTPMIP however gets much slower than the other symbolic planning algorithms as the size of the problem increases in Problem 5. This significant reduction in planning speed can be attributed to the increase in number of variables and constraints in the resulting MIP that GTPMIP solves. However, the unique capabilities of GTPMIP that are lacking in the other symbolic planning approaches are its ability to account for numerical constraints and optimize for numerical objective functions. This is demonstrated in the next experiment.

5.6.2 2D Warehouse package rearrangement Problem

Next, we evaluate GTPMIP on a series of 5 tasks to evaluate its ability to perform optimal task planning under numerical constraints. Each task is setup with a virtual robot in a 2D warehouse simulator. For each Warehouse package rearrangement

Task	**Constraints**
Task 1	$x = 0.0 \ \cap \ 0.0 \leq y \leq 4.0$ $0.0 \leq x \leq 4.0 \ \cap \ y = 0.0$ $x = 4.0 \ \cap \ 0.0 \leq y \leq 4.0$
Task 2	$x = 0.0 \ \cap \ y = 5.0$ $x = 6.0 \ \cap \ y = 5.0$ $4.5 \leq y + 1.6x \leq 5.0 \ \cap \ \ldots$ $0.0 \leq x \leq 3.0 \ \cap \ 0.0 \leq y \leq 4.0$ $-5.0 \leq y - 1.6x \leq -4.5 \ \cap \ \ldots$ $3.0 \leq x \leq 6.0 \ \cap \ 0.0 \leq x \leq 4.0$
Task 3	$x = 3.0 \ \cap \ 0.0 \leq y \leq 6.0$
Task 4	$x = 0 \ \cap \ 0.0 \leq y \leq 4.0$ $0.0 \leq x \leq 3.0 \ \cap \ y = 4.0$ $0.0 \leq 3.0 \ \cap \ y = 0.0$
Task 5	$x = 0.0 \ \cap \ 0.0 \leq y \leq 5.0$ $0.0 \leq x \leq 3.0 \ \cap \ y = 4.0$ $x = 3.0 \ \cap \ 0.0 \leq y \leq 5.0$

Table 5.2: The set of geometric constraints on package placements for each of the Warehouse package rearrangement tasks.

problem, the goal is to plan for the optimal action sequence with optimal continuous parameters that rearrange the packages by satisfying a specific set of linear geometric constraints on package placements in SE(2) space. The set of constraints for the five tasks are listed in Table 5.2.

We evaluate the time to build the Hybrid Funnel Graph as well as the time to solve the resulting MICP for each of these tasks. Quantitative experimental results for each task are presented in Table 5.3, with the corresponding qualitative results shown in Figure 5.4. The experiments were run in `westbrick.jl`, a 2D package rearrangement simulator we developed.

Videos of the robot executing the plans generated for each task can be found on the project's webpage at this url: https://adubredu.github.io/gtpmip

Task	HFG Building Time(s)	MICP Solving Time(s)
Task 1	21.12 ± 1.230	0.26 ± 0.100
Task 2	87.41 ± 10.160	0.60 ± 0.084
Task 3	5.20 ± 0.140	0.12 ± 0.004
Task 4	29.92 ± 0.450	0.31 ± 0.044
Task 5	32.06 ± 0.330	0.30 ± 0.046

Table 5.3: Times (in seconds) for building the Hybrid Funnel Graphs(HFG) and for solving the resulting Mixed Integer Convex Program for each of the tasks.

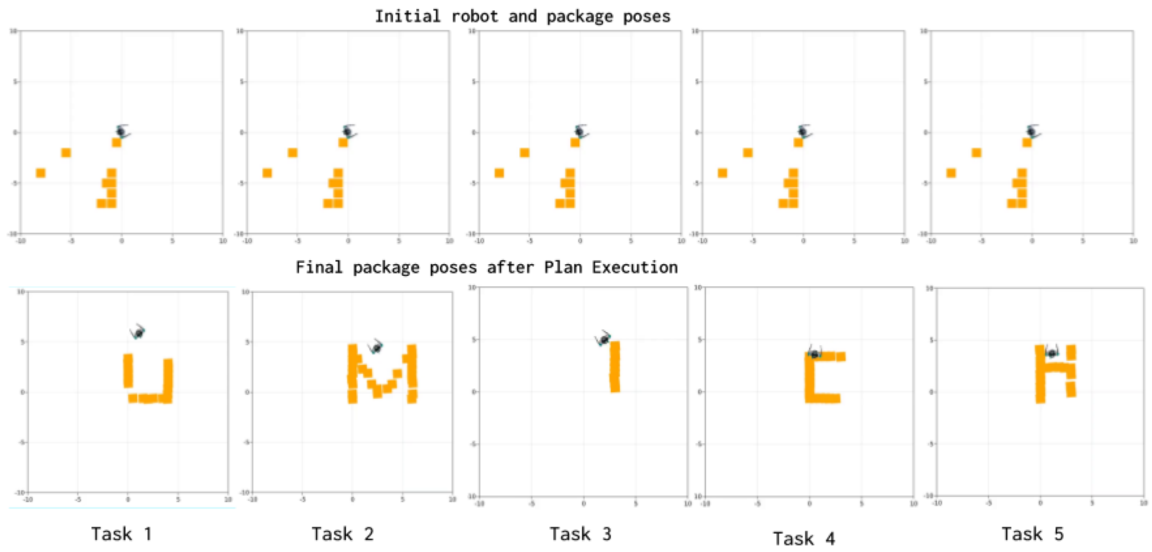


Figure 5.4: Qualitative results from the execution of plans generated by solving the package rearrangement problems in Tasks 1-5 with GTPMIP

5.6.3 3D Box Rerarrangement

5.6.3.1 Problem Description

We extend the package rearrangement problem to a 3D environment where the task is to plan an optimal action sequence for a simulated Digit humanoid robot to rearrange a pile of boxes of different masses and sizes into a pre-specified goal configuration in simulation as depicted in Figure 5.5. As this plan is to be executed by a humanoid robot, the GTPMIP problem for this task is subjected to kinematic constraints on the robot's center of mass, stance pose, end-effector position and static balance constraint.

The center of mass constraint, expressed in Equation 5.11 is a quadratic constraint that constrains the center of mass 3D position x_{com} to a desired offset δ_{com} relative to the 3D position x_{box} of the box being lifted by the robot.

$$\begin{aligned} (x_{com} - x_{box})^\top Q_{com}(x_{com} - x_{box}) &\leq \delta_{com} \\ x_{com}, x_{box}, \delta_{com} &\in \mathbb{R}^3 \end{aligned} \tag{5.11}$$

where Q_{com} is a 3x3 diagonal matrix of weights. We set Q_{com} to the identity matrix in the experiments.

Similarly, the stance constraint, expressed in Equation 5.12 is a quadratic constraint on the $\mathbb{SE}2$ pose x_{stance} of the feet of the humanoid robot to the $\mathbb{SE}2$ pose x_{box2D} of the box to be lifted by the robot. Since both feet must be in contact with the ground when the robot lifts the box, we only constrain the $\mathbb{SE}2$ pose of the robot's feet. i.e. the x, y and yaw components of the pose of the feet.

$$\begin{aligned} (x_{stance} - x_{box})^\top Q_{stance}(x_{stance} - x_{box}) &\leq \delta_{stance} \\ x_{stance}, x_{box}, \delta_{stance} &\in \mathbb{SE}2 \end{aligned} \tag{5.12}$$

where Q_{stance} is a 3x3 diagonal matrix of weights. We set Q_{stance} to the identity matrix in the experiments.

The end-effector position constraints are also quadratic constraints on the positions of the wrists of the robot with respect to the box to be lifted. The desired offset δ_{ee} is a user defined parameter that represents how close the lifted box should be from the torso of Digit. For the lighter boxes, the magnitude of δ_{ee} is large because Digit can afford to hold them without significantly changing the position of the total center of mass of the robot and the box and cause an imbalance. However, for heavier objects, the magnitude of δ_{ee} is small to ensure that Digit holds the box closer to its torso and maintains balance as it lifts and transports the box.

The final constraint is the static balance constraint which is a constraint on the center of mass and the stance pose of Digit. This constraint ensures that the horizontal projection of the center of mass remains within the convex hull of Digit's feet while it lifts the box. This is the condition for static stability of a bipedal humanoid robot.

Combining these quadratic kinematic constraints with the integer constraints of the planning problem results in a Mixed Integer Quadratically-Constrained Quadratic Program (MIQCQP). As with the all experiments in this chapter, we solve this MIQCQP using the Gurobi Mathematical Program solver.

5.6.3.2 Implementation

We used the MuJoCo Physics Simulation software[94] to simulate the Digit humanoid robot as well as the box rearrangement environment setup. Given the plan output from GTPMIP, we use the ALIP walking controller [38] to generate foot placement locations to computed stance poses from the plan and the Kinodynamic Fabrics whole-body controller [4] to track the foot placements from ALIP and the center of mass and posture parameters from the GTPMIP plan.

Problem	Num. Boxes	HFG Building Time(s)	MIQCQP Solving Time(s)
Problem 1	6	0.33	0.095
Problem 2	6	0.18	0.103
Problem 3	8	3.47	6.167
Problem 4	8	2.52	4.700
Problem 5	9	5.99	10.315

Table 5.4: Times (in seconds) for building the Hybrid Funnel Graphs(HFG) and for solving the resulting Mixed Integer Quadratically-Constrained Quadratic Program for each of the 3D Box Rearrangement problems.

5.6.3.3 Results

We formulate 5 different constrained box rearrangement problems as described above and solve them with GTPMIP. Figure 5.5 shows the initial and goal configuration of the boxes in each of the constrained box rearrangement problems. Table 5.4 presents the Hybrid Funnel Graph build times and the corresponding MIQCQP solve times for the various problems.

The experimental results show a general increase in Hybrid Funnel Graph building time and Problem solving time with increasing number of boxes. As the number of boxes increases, the total number of actions that could be taken at any time by the robot also increases. This leads to an increase in the number of integer and continuous constraints in the planning problem and by extension, an increase in the time it takes the solver to find an optimal solution. Figure 5.6 shows a step-by-step execution of grounded actions in the plan generated by GTPMIP for Problem 2.

5.6.4 Mobile Manipulation tasks with Physical Robot

Finally, we employ GTPMIP in planning for optimal constrained tasks in the real world. We use the Digit [81] humanoid robot platform to execute output plans. We focus on 2 main tasks; the shelf-stocking task (pictured in Figure 5.7) and the table serving task.

The shelf-stocking task requires that the robot stock a shelf with a predefined set of grocery items at specific positions on the shelf. The table serving task requires that the robot collects a specified set of grocery items from a shelf and distributes them to a dinner table in predefined desired configurations.

The kinematic constraints we consider in these tasks are the robot stance pose constraint and the grasp angle constraint. The robot stance pose constraint constrains the robot's standing pose to a desired region in SE(2) space from which the object to be grasped is kinematically reachable by the robot. The grasp angle constraint ensures that the angle of approach of the robot's grippers results in a stable grasp.

Video demonstrations of the robot performing all these tasks can be found on the project's website at this url: https://adubredu.github.io/gtpmip

5.7 Discussion

5.7.1 Comparison with other approaches

Ultimately, GTPMIP seeks to solve the problem of task planning with continuous constraints. The output of GTPMIP is a grounded plan made up of actions with continuous parameters. These actions can then be executed by a Robot equipped with motion controller that can generate motor commands given the actions. Task and Motion planning (TAMP) approaches [88, 35] solve a slightly different problem albeit generating a similar kind of output as GTPMIP. They interleave symbolic task planning with conditional sampling of continuous parameters for actions generated from task planning. These sampled continuous parameters are only feasible with no guarantee of optimality. As such, TAMP is restrained to only solving tasks with symbolic goals and cannot solve tasks with a continuous objective and continuous explicit constraints.

Logic Geometric Programming (LGP) [96], proposed by Marc Toussaint, is an alternative approach for solving TAMP problems with continuous objectives. LGP formulates the TAMP problem as a nonlinear optimization problem where symbolic actions are used to formulate constraints over continuous parameters. LGP solves the nonlinear optimization problem in 3 stages, with the first stage being an optimization of the final state of the continuous parameters given a symbolic plan planned separately using Monte Carlo Tree Search and the final stage being an optimization over the full continuous parameters of the entire plan. Even though this layered approach allows LGP to optimize for continuous constraints and objectives, this separation of symbolic planning and continuous parameter search loses the tight coupling of symbolic search and continuous parameter search that TAMP approaches possess. As such, LGP poses a risk of planning logically-consistent symbolic actions that may be geometrically infeasible. GTPMIP, in contrast, formulates the entire TAMP problem as a single Mixed Integer Program, jointly solving for both symbolic actions and continuous parameters, while optimizing a continuous task objective. This joint optimization removes the risk of geometrically-infeasible symbolic actions.

5.8 Conclusion

We tackled the problem of optimal constrained task planning by proposing an approach that encoded the entire task planning problem as a single MICP and solved it using an off-the-shelf MIP solver. We evaluated our approach on a set of optimal constrained task planning problems and demonstrated its ability to generate optimal plans including on a physical robot platform under kinematic constraints.

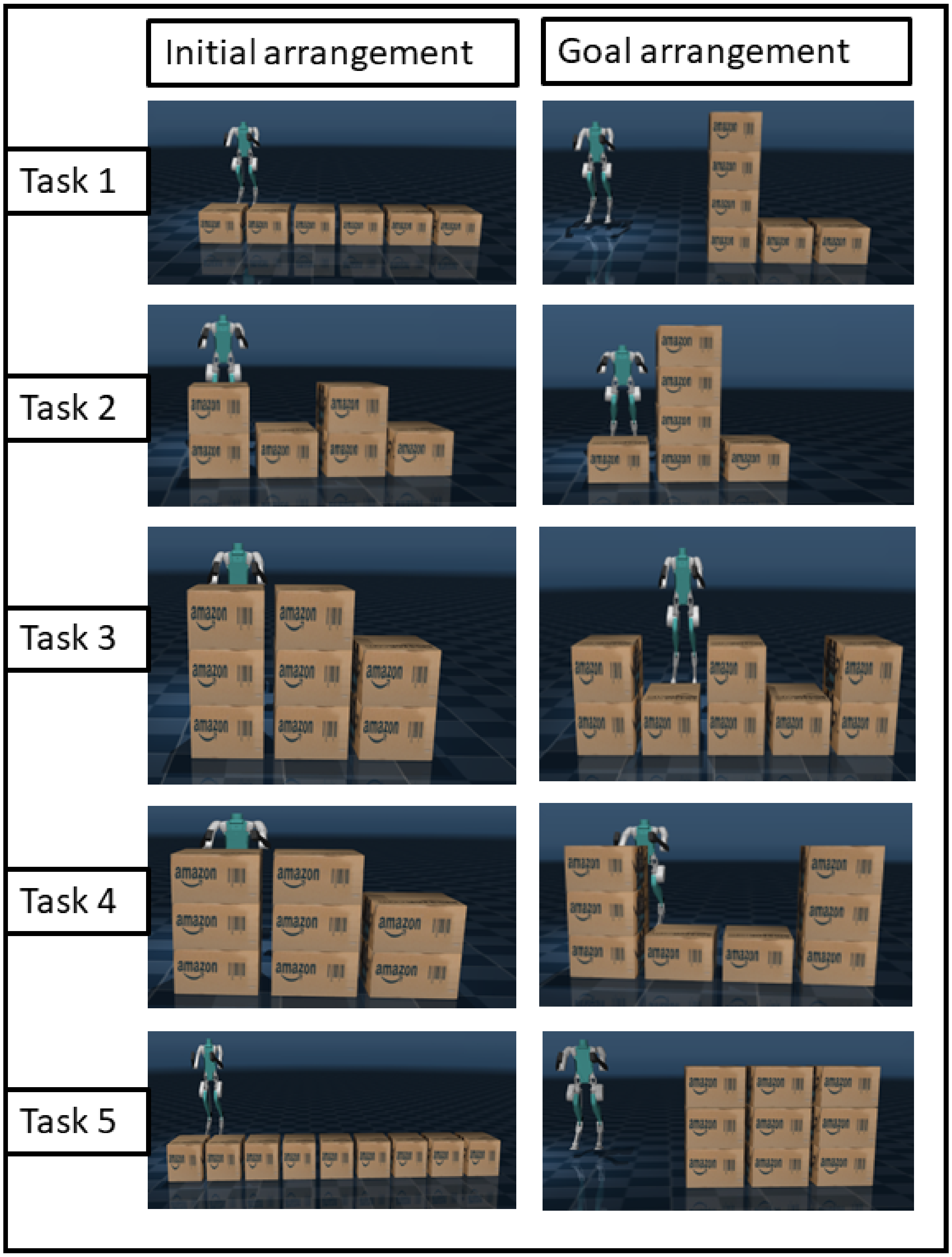


Figure 5.5: Initial and Goal box rearrangements for each of the 5 3D Box Rearrangement tasks

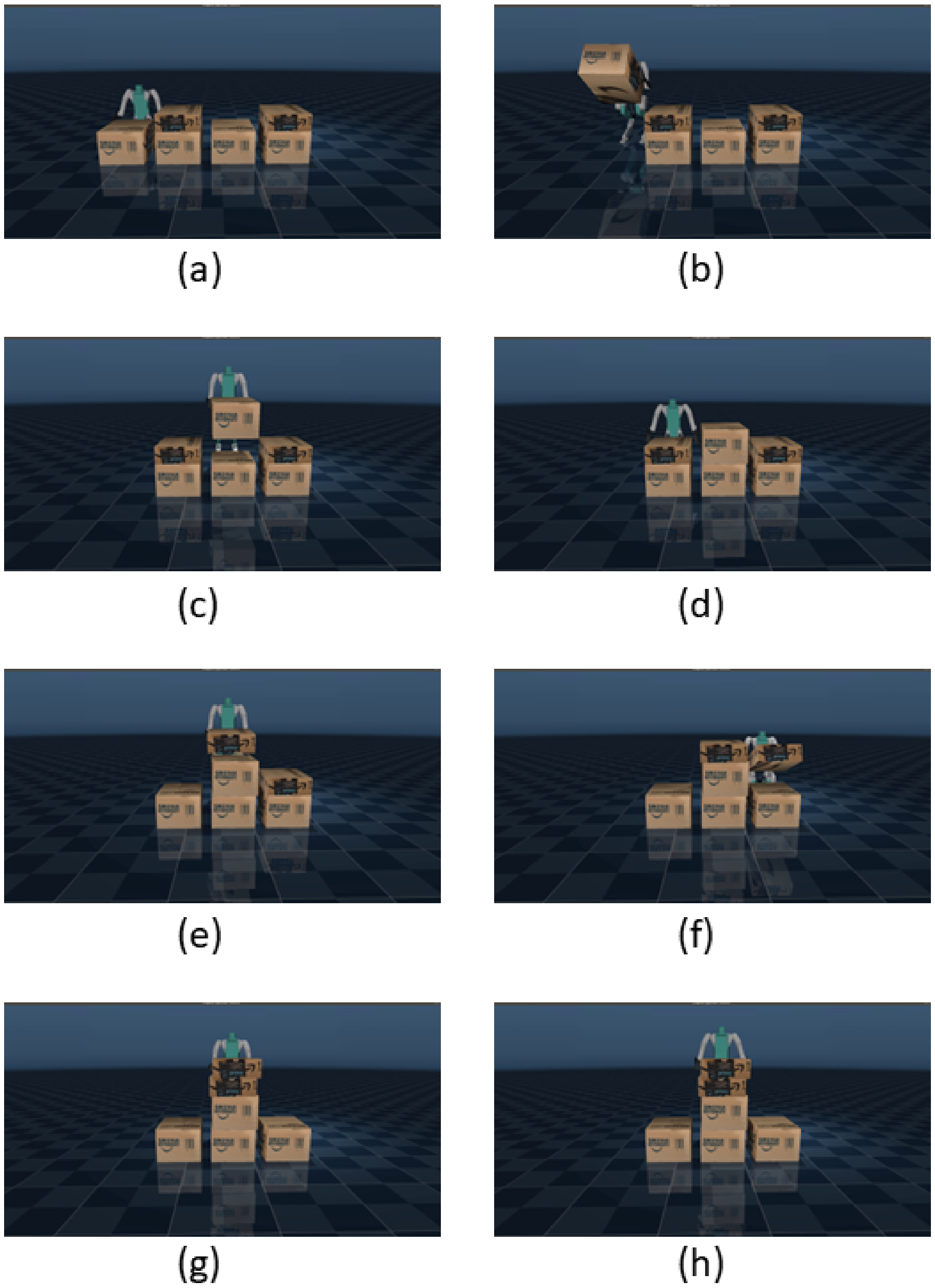


Figure 5.6: Subfigures (a) - (h) show snapshots of the Digit robot executing a GTP-MIP generated plan to rearrange large Amazon boxes.

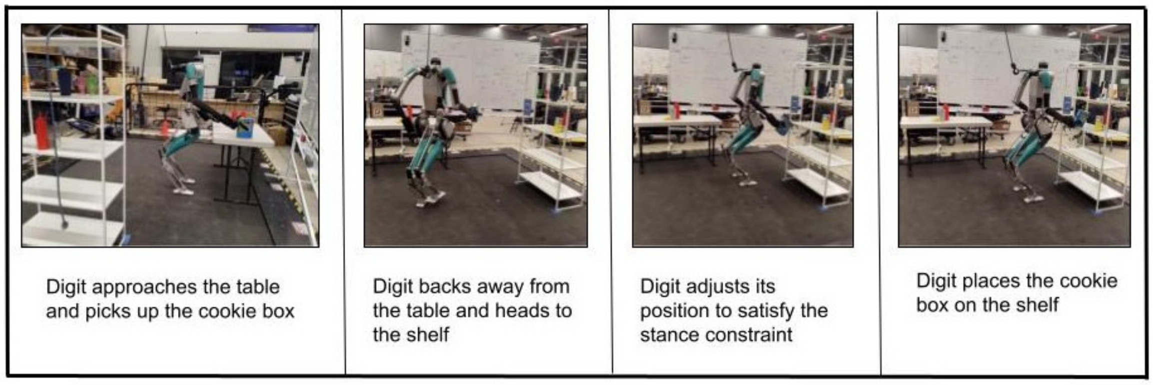

Digit approaches the table and picks up the cookie box
Digit backs away from the table and heads to the shelf
Digit adjusts its position to satisfy the stance constraint
Digit places the cookie box on the shelf

Figure 5.7: Plan execution sequence for the shelf-stocking task on a physical Digit robot

CHAPTER VI

Future Work

6.1 Relaxing GTPMIP assumptions

The primary assumption made with GTPMIP in Chapter V is that the robot had knowledge of the best constraints to satisfy and the right set of action primitives to accomplish a given task. However, robots operating in most interesting real world settings do not often have access to these capabilities.

An exciting avenue for future work would be to ease the expense of predefining constraints. Could a robot learn to derive numerical constraints from natural language or from user demonstrations? They have been a few recent efforts in this direction. A recent work by Ding et. al. [27] proposes the use Large Language Models to generate semantically-valid object arrangements and inter-object geometric relationships which are then used to describe task goals for a downstream Task and Motion Planning approach. Such learned language models can be fine-tuned to generate plausible semantic and geometric constraints for Grounded Task Planning approaches like GTPMIP.

Another potential avenue for future work would be to enable robots to autonomously learn the right set of action primitives needed to complete a given task. The action primitives used in the various works described in this dissertation were hand-designed for the various tasks. Even though the hand-designed action primitives were sufficient for the limited tasks considered in this dissertation, it is likely that a general purpose robot may encounter tasks in different environments were the hand-designed action primitives may not be sufficient at accomplishing the assigned tasks. For example, the bimanual side-ways lifting action primitive used in Chapter V to lift amazon boxes may not be appropriate for a box made up of fragile cardboard that should instead be lifted from the bottom. It will be more desirable for the robot to learn the right set of action primitives for a particular task through observing an expert demonstration. How should this robot learning problem be formulated? What is the right medium for expressing expert demonstrations? What sequence of motions from the demonstration constitutes an action primitive? Should the robot mimic the exact motions of the expert demonstration or should it be able to infer more subtle cues from the demonstration? These are certainly interesting questions to explore.

6.2 Inferring task goals from human demonstrations

A common way for humans to teach each other how to perform manual tasks is through demonstration. Kids learn how to walk, brush their teeth, and tie their shoelaces through observing how adults perform these tasks. Likewise, a seamless way to instruct robots to perform manual tasks could be through demonstration. However, the question about what exactly the robot should derive from a human demonstration still remains. Zeng et. al. in [112] proposed that the robot derive inter-object relationships from the user demonstration. These inter-object relationships are then used to formulate the task goals of a task planning problem which can be solved and executed using an approach like Lesample from Chapter III. For longer, temporally-extended demonstrations like cleaning an entire apartment or cooking or assembling car parts, deriving inter-object relationships alone may not suffice. Endowing robots with the ability to learn to predict task goals from human demonstrations, particularly from ubiquitous media like video recordings and plan grounded actions to achieve the predicted task goals would be a big step towards the development of highly capable and teachable robots.

APPENDICES

APPENDIX A

Manipulation with Digit

A.1 Gripper Hardware

We use the Digit robot extensively in robot experiments throughout this thesis. The Digit robot, pictured in Figure A.1a, is a humanoid robot with two arms and two legs. It however does not have any grippers for object pick-and-place tasks. In order to use it to perform mobile manipulation tasks that involved pick-and-place operations, we designed specialized claw grippers for Digit. The claw grippers, pictured in Figure A.1b, were actuated by two high-torque, 2.4 Newton-meter servomotors which run at 9v each. The servomotors were controlled using a tiny micro-controller encased in the gripper assembly. The micro-controller had a Radio-Frequency receiver that intercepted transmitted signals from a transmitter module connected via a serial port to Digit's central computer. This wireless setup ensured that there were no dangling wires around the torso and arms of the robot, as wires could end up tangling as Digit's walked about and swung its arms.

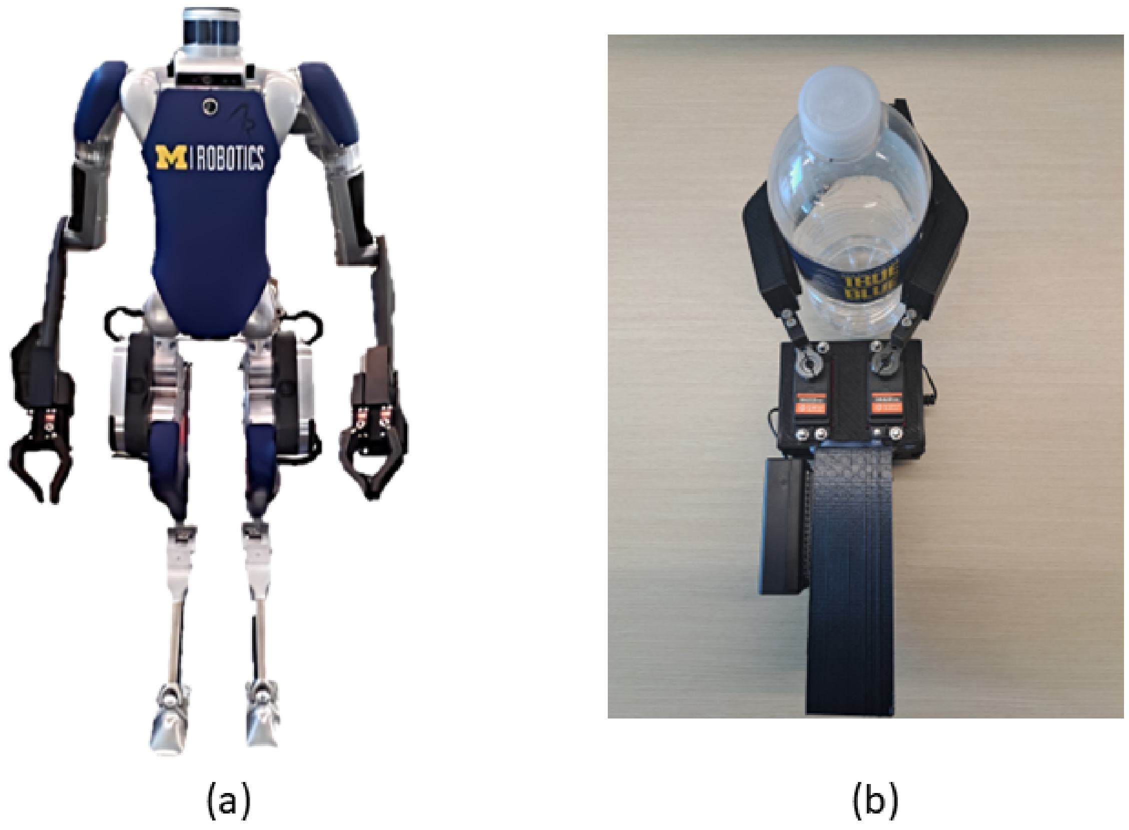


Figure A.1: (a) The Digit Bipedal Humanoid Robot with custom claw grippers attached (b) Custom claw grippers grasping a plastic water bottle

A.2 Gripper Software

The actuation of the gripper was controlled through code running on Digit's central computer. To close or open the gripper, the code sends a stream of bytes via the serial port to the transmitter module's micro-controller. The micro-controller then encodes the input bytes and transmits them wirelessly over a 2.4GHz radio frequency channel to the receiver module encased in each gripper assembly attached to Digit's wrists. The micro-controller of the receiver module interprets the received bytes and sends the corresponding Pulse-width modulation signal to the servomotors of the claw gripper to drive them to the specified positions.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Bernardo Aceituno-Cabezas and Alberto Rodriguez. A global quasi-dynamic model for contact-trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2020.

[2] Alphonsus Adu-Bredu, Nikhil Devraj, and Odest Chadwicke Jenkins. Optimal constrained task planning as mixed integer programming. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[3] Alphonsus Adu-Bredu, Nikhil Devraj, Pin-Han Lin, Zhen Zeng, and Odest Chadwicke Jenkins. Probabilistic inference in planning for partially observable long horizon problems. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3154–3161. IEEE, 2021.

[4] Alphonsus Adu-Bredu, Grant Gibson, and Jessy Grizzle. Exploring kinodynamic fabrics for reactive whole-body control of underactuated humanoid robots. 2023.

[5] Alphonsus Adu-Bredu, Zhen Zeng, Neha Pusalkar, and Odest Chadwicke Jenkins. Elephants don't pack groceries: Robot task planning for low entropy belief states. *IEEE Robotics and Automation Letters*, 7(1):25–32, 2022.

[6] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[7] Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan. 2020.

[8] Hagai Attias. Planning by probabilistic inference. In *AISTATS*. Citeseer, 2003.

[9] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *ARTIFICIAL INTELLIGENCE*, 90:1636–1642, 1995.

[10] Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 16(10):485–488, 2012.

[11] Rodney A Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15, 1990.

[12] Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159, 1991.

[13] Brian G Buss, Alireza Ramezani, Kaveh Akbari Hamed, Brent A Griffin, Kevin S Galloway, and Jessy W Grizzle. Preliminary walking experiments with underactuated 3d bipedal robot marlo. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2529–2536. IEEE, 2014.

[14] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[15] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3d mapping. In *Robotics: Science and Systems*, volume 11, pages 3–12, 2015.

[16] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[17] Christine Chevallereau, Gabriel Abba, Franck Plestan, Eric Westervelt, Carlos Canudas de Wit, Jessy Grizzle, et al. Rabbit: A testbed for advanced control theory. *IEEE Control Systems Magazine*, 23(5):57–79, 2003.

[18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

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

[20] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[21] Rina Dechter. Constraint networks. 1992.

[22] Edgar Degas. Edgar degas: The complete works. https://www.edgar-degas.org/.

[23] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS international conference on humanoid robots*, pages 279–286. IEEE, 2014.

[24] K. Desingh, S. Lu, A. Opipari, and O. C. Jenkins. Factored pose estimation of articulated objects using efficient nonparametric belief propagation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7221–7227, 2019.

[25] Karthik Desingh, Anthony Opipari, and Odest Chadwicke Jenkins. Pull message passing for nonparametric belief propagation. *arXiv preprint arXiv:1807.10487*, 2018.

[26] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs.* PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[27] Yan Ding and Xiaohan Zhang. Leveraging commonsense knowledge from large language models for task and motion planning. 2023.

[28] Stefan Edelkamp and Peter Kissmann. Optimal symbolic planning with action costs and preferences. In *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer, 2009.

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

[30] Robert J Full and Daniel E Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of experimental biology*, 202(23):3325–3332, 1999.

[31] Caelan Reed Garrett. Pybullet planning. https://pypi.org/project/pybullet-planning/, 2018.

[32] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1), 2021.

[33] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.

[34] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[35] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*, 37(13-14):1796–1825, 2018.

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

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

[38] Yukai Gong and Jessy W Grizzle. Zero dynamics, pendulum models, and angular momentum in feedback control of bipedal locomotion. *Journal of Dynamic Systems, Measurement, and Control*, 144(12):121006, 2022.

[39] Jessy W Grizzle, Gabriel Abba, and Franck Plestan. Asymptotically stable walking for biped robots: Analysis via systems with impulse effects. *IEEE Transactions on automatic control*, 46(1):51–64, 2001.

[40] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984.

[41] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. `https://www.gurobi.com`, 2022.

[42] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel. Modular task and motion planning in belief space. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4991–4998, 2015.

[43] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

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

[45] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246, 2006.

[46] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

[47] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[48] François Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. *arXiv preprint arXiv:1611.08268*, 2016.

[49] Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based robotic information gathering algorithms. *The International Journal of Robotics Research*, 33(9):1271–1287, 2014.

[50] Philip Holmes, Robert J Full, Dan Koditschek, and John Guckenheimer. The dynamics of legged locomotion: Models, analyses, and challenges. *SIAM review*, 48(2):207–304, 2006.

[51] IBM. Ibm ilog cplex optimizer. https://www.mosek.com/.

[52] Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana Nau. Optimal planning with global numerical state constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, pages 145–153, 2014.

[53] Matthew Jordan and Alejandro Perez. Optimal bidirectional rapidly-exploring random trees. 2013.

[54] Julia Mathematical Programming. Jump nonlinear integer program solver. https://github.com/lanl-ansi/Juniper.jl.

[55] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

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

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

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

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

[60] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–10, 2017.

[61] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas De Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. Design of a momentum-based control framework and application to the humanoid robot atlas. *International Journal of Humanoid Robotics*, 13(01):1650007, 2016.

[62] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[63] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40(3):429–455, 2016.

[64] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 33(14):1726–1747, 2014.

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

[66] Jon Lee and Sven Leyffer. *Mixed integer nonlinear programming*, volume 154. Springer Science & Business Media, 2011.

[67] Hui X Li and Brian C Williams. Generative planning for hybrid systems based on flow tubes. In *ICAPS*, pages 206–213, 2008.

[68] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[69] Iain Little, Sylvie Thiebaux, et al. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[70] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3684–3691, 2014.

[71] Adrienne Mayor. Gods and robots. In *Gods and Robots*. Princeton University Press, 2018.

[72] Andrea Montanari, Federico Ricci-Tersenghi, and Guilhem Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. *arXiv preprint arXiv:0709.1667*, 2007.

[73] T. K. Moon and J. H. Gunther. Multiple constraint satisfaction by belief propagation: An example using sudoku. In *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, pages 122–126, 2006.

[74] MOSEK ApS. Mosek solver. https://www.mosek.com/.

[75] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers*, 26(09):917–922, 1977.

[76] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[77] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. 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)*, pages 6059–6066. IEEE, 2017.

[78] C. Phiquepal and M. Toussaint. Combined task and motion planning under partial observability: An optimization-based approach. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9000–9006, 2019.

[79] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003.

[80] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[81] Agility Robotics. Digit robot. https://www.agilityrobotics.com/meet-digit.

[82] Sebastian Pokutta et. al. Scip: Solving constraint integer programs. https://scipopt.org/.

[83] Norbert M. Seel. *Action Schemas*, pages 73–75. Springer US, Boston, MA, 2012.

[84] R. Sharma and Y. Aloimonos. Coordinated motion planning: the warehouseman's problem with constraints on free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(1):130–141, 1992.

[85] David Silver and Joel Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.

[86] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26, 2013.

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

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

[89] Anthony Stentz et al. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[90] Z. Sui, O. C. Jenkins, and K. Desingh. Axiomatic particle filtering for goal-directed robotic manipulation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4429–4436, 2015.

[91] Zhiqiang Sui, Odest Chadwicke Jenkins, and Karthik Desingh. Axiomatic particle filtering for goal-directed robotic manipulation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4429–4436. IEEE, 2015.

[92] Zhiqiang Sui, Zheming Zhou, Zhen Zeng, and Odest Chadwicke Jenkins. Sum: Sequential scene understanding and manipulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3281–3288. IEEE, 2017.

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

[94] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[95] M. Toussaint and C. Goerick. Probabilistic inference for structured planning in robotics. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3068–3073, 2007.

[96] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[97] Andrés Klee Valenzuela. *Mixed-integer convex optimization for planning aggressive motions of legged robots over rough terrain*. PhD thesis, Massachusetts Institute of Technology, 2016.

[98] Menkes HL Van Den Briel and Subbarao Kambhampati. Optiplan: Unifying ip-based and graph-based planning. *Journal of Artificial Intelligence Research*, 24:919–931, 2005.

[99] J.P. Vielma, S. Ahmed, and G.L. Nemhauser. Mixed integer linear programming formulations for probabilistic constraints. *Operations Research Letters*, 40(3):153–158, 2012.

[100] Thomas Vossen, Michael O Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in ai planning. Technical report, 1999.

[101] Fan Wang and Kris Hauser. Stable bin packing of non-convex 3d objects with a robot manipulator. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8698–8704. IEEE, 2019.

[102] Fan Wang and Kris Hauser. Robot packing with known items and nondeterministic arrival order. *IEEE Transactions on Automation Science and Engineering*, 18(4):1901–1915, 2020.

[103] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2018.

[104] Eric R Westervelt, Jessy W Grizzle, and Daniel E Koditschek. Hybrid zero dynamics of planar biped walkers. *IEEE transactions on automatic control*, 48(1):42–56, 2003.

[105] 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.

[106] Chak-Kuen Wong and Malcolm C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

[107] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 2814–2819. IEEE, 2013.

[108] Yong Wu, Wenkai Li, Mark Goh, and Robert De Souza. Three-dimensional bin packing problem with variable bin height. *European journal of operational research*, 202(2):347–355, 2010.

[109] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007.

[110] Håkan LS Younes and Michael L Littman. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2:99, 2004.

[111] Zhen Zeng, Yunwen Zhou, Odest Chadwicke Jenkins, and Karthik Desingh. Semantic mapping with simultaneous object detection and localization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 911–918. IEEE, 2018.

[112] Zhen Zeng, Zheming Zhou, Zhiqiang Sui, and Odest Chadwicke Jenkins. Semantic robot programming for goal-directed manipulation in cluttered scenes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7462–7469, 2018.

[113] Zheming Zhou, Tianyang Pan, Shiyu Wu, Haonan Chang, and Odest Chadwicke Jenkins. Glassloc: plenoptic grasp pose detection in transparent clutter. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4776–4783. IEEE, 2019.

[114] Zheming Zhou, Zhiqiang Sui, and Odest Chadwicke Jenkins. Plenoptic monte carlo object localization for robot grasping under layered translucency. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.