AUTONOMOUS ENVIRONMENT MANIPULATION TO FACILITATE TASK COMPLETION

A Thesis
Presented to
The Academic Faculty

by

Martin Levihn

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

School of Interactive Computing
Georgia Institute of Technology
May 2015

AUTONOMOUS ENVIRONMENT MANIPULATION TO FACILITATE TASK COMPLETION

This thesis is dedicated to the memory of Mike Stilman, whose encouragement, support and enthusiasm will never be forgotten.

ACKNOWLEDGEMENTS

I am very grateful for having had the opportunity to be advised by Mike Stilman for almost five years. His energy and enthusiasm for research made me stay at Georgia Tech and pursue a PhD despite my original plans of returning to Germany after my masters. Thanks to Mike's easy-going personality, he always felt more like a friend than an advisor and, from the very beginning, he put a lot of trust in me. During my first year as a PhD student, he supported me in going to Japan for a couple months to experience a different culture and research environment. In fact, he even came to Japan for a week while Tobias Kunz and I were there and we all just traveled through Japan and gave talks at various labs. I have learned a lot from him throughout the years and I will be forever grateful.

I would like to thank the College of Computing at Georgia Tech and specially Henrik Christensen and Annie Anton for their unlimited support after the passing of Mike. Without hesitation, Henrik offered to advise me throughout the rest of my PhD and has provided me with invaluable insights both academically and in regards to career choices. Annie made it to her priority to ensure that all of Mike's students and colleagues had all the necessary guidance. Both Henrik and Annie were always available for us and their support is the reason I am writing this thesis. I would further like to thank all the faculty and staff at Georgia Tech that worked hard in the background to ensure that all of Mike's students were taken care of.

I am extremely grateful to the members of my committee. Charles Isbell provided me with very valuable feedback about publishing with the Machine Learning community and graduate student life in general. Magnus Egerstedt helped me to see the bigger picture of my research and clearly articulate the properties and limitations of the approaches presented in this thesis. Frank Dellaert was always available for discussions and represented an invaluable resource for both scientific and carrier insights. I also especially thank Tomás Lozano-Pérez and Leslie Pack Kaelbling. They have accepted me into their lab at MIT as a visiting graduate student and treated me like one of their regular students. I ended up staying for almost 8 months at their lab and learned a tremendous amount. I will never forget the countless hours that Leslie and Tomas were actively hacking on the robot with me, even on Sundays. Their enthusiasm about research and robotics is unparalleled. Chapter 4 is the result of this collaboration.

In addition, I am very thankful to Koichi Nishiwaki, Satoshi Kagami and Takeo Igarashi all of which hosted me at labs in Japan. Nishiwaki-san and Kagami-san hosted me during my time at the Digital Human Research Center and were unbelievable patient with me constantly breaking their robots. The collaboration resulted in the first real humanoid robot deciding to build itself a bridge and is discussed in Chapter 5. Igarashi-sensei hosted me at the Design Interface Project and provided me with great insights into multi-robot systems. Our collaboration resulted in a multi-robot paper that we presented at IROS.

I am grateful for the friends I made both at Georgia Tech and MIT. Jonathan Scholz was an invaluable source, academically and personally. We had many very insightful discussions and co-authored numerous papers throughout the years. We spend endless hours at coffee shops in and around Atlanta debating approaches and solutions; not always to the pleasure of the other guests in the coffee shops. We also had numerous BBQs and trips to recharge the batteries and keep one sane throughout the years. Baris Akgun was always available when needed, whether it was code debugging, a coffee break, or a road trip. His advise giving for both professional and personal issues made him an invaluable resource. Michael ”Misha” Novitzky brought personality to the group. Something that everyone that we passed during our morning runs can attest to. No matter the distance we were running and how exhausted we were, Misha would always greet every person we passed. Misha also let me workout and practice martial arts at his Dojo, ensuring that I get regular concussions to reset some wrong algorithm approaches. I would also like to thank Jonathan Gladin, Jaswanth Sreeram, Sasha Lambert, Rowland O'Flaherty, Heni Ben Amor, Neil Dantam, Kaushik Subramanian, Ashley Edwards, Himanshu Sahni, Pushkar Kolhe and Tushar Kumar for their friendships throughout the years. At MIT, I would like to thank all of the LIS group members and especially my officemates Jenny Berry, Alejandro Perez and Sam Davies for their help and support during my stay.

I could not have accomplished this journey without constant and unlimited support from my family. My parents backed me in every decision I made during my graduate studies. Whether it was staying in the US for a PhD, buying an old Buick or going to Japan for a couple months, they always provided both encouragement and a safety net. I would not have been able to pull through without this support.

Finally, I cannot express in words the gratitude I have towards my girlfriend Katharina Wunderlich. We dated throughout my entire time as a PhD student and despite large distances separating us, she was always there when needed. Her willingness to stay in the relationship despite entire continents separating us for months at a time and her calming manner whenever things got heated enabled me to work on this thesis throughout the years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A robot should be able to autonomously *modify and utilize* its environment to assist its task completion. While mobile manipulators and humanoid robots have both locomotion and manipulation capabilities, planning systems typically just consider one or the other. In traditional motion planning the planner attempts to find a collision free path from the robot's current configuration to some goal configuration. In general, this process entirely ignores the fact that the robot has manipulation capabilities. This is in contrast to how humans naturally act - utilizing their manipulation capabilities to modify the environment to assist locomotion. If necessary, humans do not hesitate to move objects, such as chairs, out of their way or even place an object, such as a board, on the ground to reach an otherwise unreachable goal. We argue that robots should demonstrate similar behavior. Robots should use their manipulation capabilities to *move* or even *use* environment objects. This thesis aims at bringing robots closer to such capabilities.

There are two primary challenges in developing practical systems that allow a real robotic system to tightly couple its manipulation and locomotion capabilities: the inevitable inaccuracies in perception as well as actuation that occur on physical systems, and the exponential size of the search space. To address these challenges, this thesis first extends the previously introduced domain of Navigation Among Movable Obstacles (NAMO), which allows a robot to move obstacles out of its way. We extend the NAMO domain to handle the underlying issue of uncertainty. In fact, this thesis introduces the first NAMO framework that allows a real robotic systems to consider sensing and action uncertainties while reasoning about moving objects out of the way. However, the NAMO domain itself has the shortcoming that it only considers a robot's manipulation capabilities in the context of clearing a path. This thesis therefore also generalizes the NAMO domain itself to the Navigation Using Manipulable Obstacles (NUMO) domain.

The NUMO domain enables a robot to more generally consider the coupling between manipulation and locomotion capabilities and supports reasoning about *using* objects in the environment. This thesis shows the relationship between the NAMO and NUMO domain, both in terms of complexity as well as solution approaches, and presents multiple realizations of the NUMO domain. The first NUMO realization enables a robot to use its manipulation capabilities to assist its locomotion by changing the geometry of the environment for scenarios in which obstructions can be overcome through the usage of a single object. The system led a real humanoid robot to autonomously build itself a bridge to cross a gap and a stair step to get on a platform. A second NUMO realization then introduces reasoning about force constraints using knowledge about the mechanical advantages of a lever and battering ram. The discussed system allows a robot to consider increasing its effective force though the use of objects, such as utilizing a rod as a lever. Finally this thesis extends the NUMO framework for geometric constraints to scenarios in which the robot is faced with a substantial lack of initial state information and only has access to onboard sensing.

In summary, this thesis enables robots to autonomously modify their environment to achieve task completion in the presence of lack of support for mobility, the need to increase force capabilities and partial knowledge.

CHAPTER I

INTRODUCTION

The goal of this thesis is to enable robots to autonomously modify their environment to achieve a task. In traditional robot motion planning systems, the robot sees environment objects as obstacles. We propose that the robot should also see them as opportunities. Each manipulable object is an opportunity for the robot to change its environment to its advantage, from moving them out of the way to even utilizing them as tools. For a robot, not all environment objects have to be obstacles; they can also be a means to an end.

As a first step towards this goal, we extend the previously introduced domain of Navigation Among Movable Obstacles (NAMO) which enables robots to reason about moving objects out of the way. While previous work introduced important concepts to make the NAMO domain tractable, the majority of existing methods assume full world knowledge and perfect action execution by the robot. In contrast, real world systems can typically only perceive the environment through limited sensory measurements, resulting in state and action uncertainty. In order to move NAMO systems closer to real world applicability, we extend the NAMO domain to handle state and action uncertainties and present a real robot implementation operating amid these conditions.

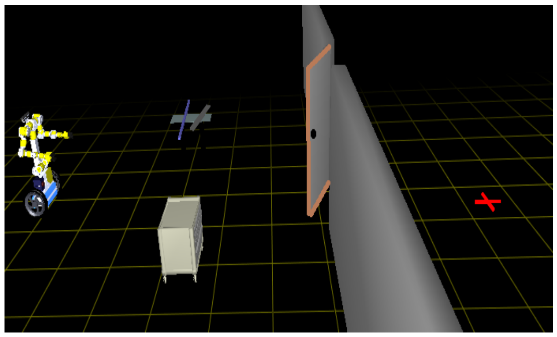
While this extension is a crucial step towards real robotic systems that can make progress towards a goal even if they are blocked in by obstacles, it does not allow robots to reach a goal that is outside of their inherent physical limitations. For example, if a robot is disconnected from the goal by a large gap, any reasoning about moving objects out of the way will not allow the robot to find a valid plan to the goal. Instead, the robot needs to find ways of using environment objects to cross the gap, such as placing a board over the gap. These kind of behavior patterns require the ability to reason about environment objects as tools. In the second part of this thesis, we therefore generalize the concept of Navigation *Among* Movable Obstacles to Navigation *Using* Manipulable Obstacles (NUMO). While in the NAMO domain the robot only reasons about topological environment changes by moving objects out of the way, the NUMO domain allows the robot to also reason about *using* manipulable objects to modify the environment to its advantage. This generalization allows robots to solve previously unsolvable problems and brings them closer to the tool orientated behavior characteristic of humans.

The remainder of this chapter describes the motivation behind this thesis and the challenges that need to be overcome to develop practical systems. A brief overview of the structure of this document is provided at the end of the chapter.

1.1 Motivation

Future rescue robots that save humans from disasters such as floods and earthquakes will be required to reason about objects in their environment. Traditional motion planning algorithms search for collision-free paths from the start to the goal [52]. This is not sufficient when flood waters have caused furniture to float and collapse, leaving no open path to the victims. Instead, the robot must quickly decide which obstacles must be moved or potentially even stepped on to reach the goal. It must choose where to move objects, whether to to create a traversable path or clear a path, and compute valid motion plans that integrate navigation and manipulation.

The need for such capabilities became evident during the recent crisis at the Fukushima Daiichi nuclear power plant where hydrogen-air chemical explosions critically damaged the containment buildings [1]. To observe the damage and to reach crucial instruments, it was critical to enter the damaged buildings. As the radiation levels were too high for workers to enter, the nuclear power plant operators turned to robotics [85]. However, current robot technology proved to be limited [85]. The robots required constant teleoperation, which in turn required the operators to be within close proximity to the power plant at all times


(a) The door is jammed and cannot be moved by the robot directly.

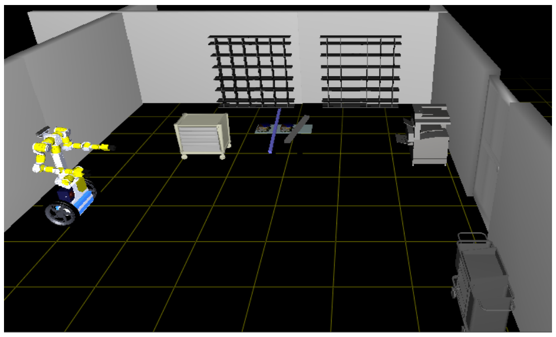
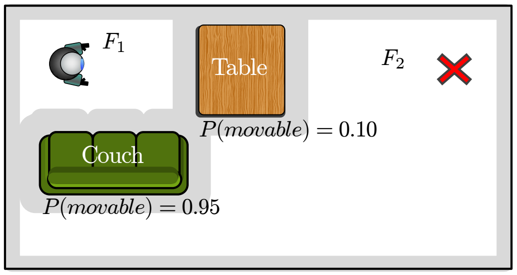

(b) The door is locked and opens outwards.

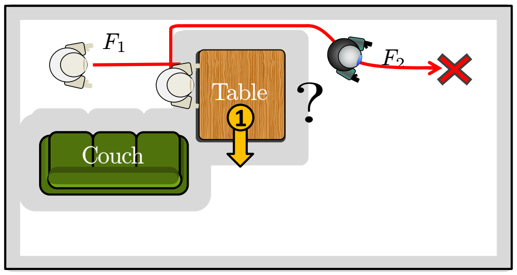
Figure 1: Example setup. In both examples the robot is tasked with exiting the room.

in full protection suits, increasing the difficulty of operating the robots teleoperation command terminal while exposing the workers to radiation [85]. In addition, if communication between the operation terminal and the robot broke, the robot would be lost and the task could not be completed. If the robots would have been able to operate more autonomously and reach target areas without constant supervision and teleoperation, radiation exposure of the workers could have been lowered and task completion rate increased. Being able to autonomously decide to move or even use environment objects is a crucial step in this direction.

To understand the implications of such capabilities, consider the scenario visualized in Fig. 1(a) in which a robot is trapped in a room with a jammed door. If unjamming the door requires more force than the robot can exert on its own, current systems would just directly declare failure, entirely ignoring the scope of the robots manipulation capabilities and the objects in the environment. This puts the task completion at risk, even if escaping the room is vital and possible. In contrast, for humans, the natural approach is to reason about different ways to increase their effective force. Utilizing knowledge about *mechanical advantage* afforded by possible combinations of objects, humans find creative solutions such as using a rod and an office cart to achieve leverage to pry open the door in Fig. 1(a), or use an office cart as a battering ram to break the lock of the door for the example visualized in Fig. 1(b). Robots should show similar behaviors and we argue that the capability to



(a) Configuration



(b) Solution obtained through a deterministic NAMO planner.

Figure 2: The table wheels are likely to be locked, making it impossible for the robot to move the table. In contrast to deterministic planners, our proposed framework accounts for this probability.

jointly reason about manipulation and locomotion abilities is on the critical path to truly intelligent, multi-purpose robots. In fact, the ability to use objects in the environment to achieve goals is considered one of the hallmarks of intelligence in humans, primates and even some birds [44, 86, 105]. This thesis aims at brining robots closer to this hallmark.

1.2 Challenges

There are two primary challenges in developing practical systems that allow a real robotic system to reason about environment objects: the dimensionality of the state space and the inevitable uncertainty in state and action outcome on real systems.

1.2.1 Dimensionality

To understand the state space complexity, consider navigating in a room with manipulable obstacles such as the ones depicted in Fig. 2. If $\mathscr{C}_r \in \mathbb{R}^{d_r}$ represents the configuration of the robot, and $\mathscr{C}_o \in \mathbb{R}^{d_o}$ represents the configuration of a *single* obstacle, then the full state-space consisting of the robot and N obstacles is the *product* of these subspaces: $\mathscr{C}_r \times \mathscr{C}_{o_1} \times \mathscr{C}_{o_2} \times \ldots \times \mathscr{C}_{o_N} \in \mathbb{R}^{d_r + N d_o}$. For discretized environments with resolution r in each dimension, the number of possible states is $O(r^{d_r + N d_o})$, in other words exponential in the number of objects it contains. This complexity analysis holds for cases where the robot is only reasoning about moving objects out of the way as well as for the more general case of using environment objects. In both domains, objects need to be rearranged to change the configuration space topology.

1.2.1.1 Problem Classifications

To simplify further analyses, we now introduce a problem classification based on solution difficulty. In the following discussion we use the term *free-space* to indicate configuration space regions that allow the robot to navigate collision free between any two points within the region. Note that whether the robot is just reasoning about moving objects or also considers using objects, the robot can complete its overall task if and only if it manages to get access to the free-space region containing the goal. Our classification is a generalization of the NAMO domain classification presented in [95] and defines the following problem instances:

I_k Independent subproblem domains where free-space regions can be connected independently by interacting with at most k objects per subproblem.

NI Not-independent domains that cannot be subdivided into indepdent subproblems.

UL Usage-limited problems that require the robot to interact with any object at most once.

UU Unlimited-usage problems in which the robot might have to interact with the same objects multiple times.

A planner typically is designed for any combination of these subclasses, *e.g.* a planner designed for I_1UU problems reasons about solving each subproblem independently by using any of the objects in the environment, independent of the objects prior usages. Note that the main difference between this problem classification and the one presented in [95] is the focus on object usage.

1.2.2 Uncertainty

Prior work on autonomous configuration space topology changes, *e.g.* [92, 93, 104] for the NAMO domain, has focused primarily on handling the problem of *dimensionality*, and has not yet addressed the underlying issue of *uncertainty*. In reality, robots have incomplete world model and can only perceive the environment through limited sensory input, resulting in state and action uncertainty.

To better understand why this might be a problem, consider again the example in Fig. 2. Perhaps the robot knows that the shortest path to the goal involves moving the table, but it does not know whether all the table wheels are unlocked. How might it weigh the costs of moving the table versus the couch? How would this answer be affected if it were given only a crude action model for the dynamics of the couch? These sorts of reasoning patterns are not expressible within the framework of deterministic search, without resorting to *ad hoc* heuristics such as attempting to roll the action uncertainty into a single numerical value in the cost function. However, for the robot to execute the lowest expected cost actions, it has to explicitly reason about its degree of environment uncertainty [52].

In addition to action uncertainty, a real robot system needs to be able to handle state uncertainty. While the example in Fig. 2 incorporates the wheel state uncertainty into the action uncertainty, this is not a valuable approach for more substantial state uncertainty, such as a lack of object position information. If the robot does not actively reason about sensing actions, it is in danger of colliding with objects. In order to be deployable on real robots, which in general only possess limited onboard sensing capabilities, systems allowing the robot to move or even use environment objects need to be able to handle cases with substantial lack of initial state information.

1.3 Approach

We address these challenges in this thesis by combining domain insights with hierarchical task decomposition. First, we utilize the concept of free-space regions to construct a hierarchical Markov Decision Process (MDP) that closely resembles the real problem, but can be solved in real-time. For scenarios in which the robot needs to move single objects out of the way to connect individual free-space regions, this abstract MDP allows the robot to bias its decisions at plan time in order to compute policies that are likely to succeed. Building on these results, we then extend the hierarchical task decomposition and present a system that is applicable to environments with substantial state and action uncertainty. We achieve this by incorporating the concepts of Reconsideration and Foresight, which are frequently used by humans, into a hierarchal planning and execution architecture.

To enable the robot to also perceive environment objects as potential tools and not just obstacles, we introduce the domain of Navigation Using Manipulable Obstacles (NUMO). The NUMO domain allows the robot to *use* environment objects. To make the NUMO domain tractable, we demonstrate how search space reduction concepts that have proven viable in related domains can be transferred to the NUMO domain. First, we show how constraint relaxed planning, which lets the planner hypothesize a successful scenario, can be used to allow the system to focus on regions that are likely to contain a solution. This contribution allows a real humanoid robot to determine complex, multi-step plans, such as building a stair step or bridge before navigating to the goal, within seconds.

To allow the robot to also reason about increasing its effective force by using environment objects, we use physcial constraint propagation to effectively reduce the research space without eliminating any potential solutions. The resulting search space can efficiently be sampled and enables the robot to reason about building itself a lever-like structure to pry open a door or even using an office cart as a battering ram to break open a lock in real-time.

Finally, we combine constraint relaxed planning with execution monitoring to allow the robot to *online* reason about using environment objects as tools.

1.4 Thesis Overview

We now provide the thesis statement as well as a concise overview of the contributions and the remaining parts of this document.

1.4.1 Thesis Statement

Robots can autonomously modify their environment to achieve task completion in the presence of lack of support for mobility, the need to increase force capabilities and partial knowledge.

1.4.2 Summary of Contributions

The contributions made by this thesis are as follows:

- *NAMO-MDP* — the first decision-theoretical planning system for the NAMO domain focusing on action uncertainty. The system is applicable for scenarios in which a single object can be used to create a path between two free-space regions and presents a novel combination of hierarchical MDP decompositions and Monte Carlo Tree search methods allowing for linear time complexity for typical environments.

- *Foresight and Reconsideration in Hierarchical Planning and Execution* — a hierarchical planning and execution architecture that maintains the computational efficiency of hierarchical decomposition while improving optimality, allowing a real robot to solve NAMO domains with state and action uncertainty.

- *NUMO* — Navigation Using Manipulable Objects, a new planning domain generalizing the NAMO domain to also allow a robot to reason about *using* environment objects.

- *NUMO for Geometry Constraints* — a realization of the NUMO domain enabling a real humanoid robot to use objects in the environment to overcome geometric obstructions. The system is applicable for scenarios in which a single object can be used to overcome an obstruction.

- *NUMO for Force Constraints* — a realization of the NUMO domain that allows a robot to reason about increasing its effective force through the use of environment objects by utilizing knowledge about lever and battering ram structures.
- *NUMO with Onboard Sensing* — an extension of NUMO to cases where the robot only has access to onboard sensing information.

1.4.3 Document Outline

The remainder of this document is structured as follows:

Chapter 2 - Related Work provides and overview of related work for the fields of NAMO planning, robotic planning under uncertainty, robotic tool use and object detection.

Chapter 3 - NAMO with Action Uncertainty introduces the NAMO-MDP, a framework to efficiently compute policies for NAMO domains where the robot has access to object locations but has substantial uncertainties in the object dynamics.

Chapter 4 - NAMO with State and Action Uncertainty presents a novel method to detect partially occluded objects in 3D point clouds and a complete system using Foresight and Reconsideration in a hierarchical planning and execution framework that allows a real robot to execute NAMO with state and action uncertainty.

Chapter 5 - Navigation Using Manipulable Objects introduces the NUMO domain and presents two example systems for this new domain: A humanoid robot overcoming its locomotion limitations and a humanoid robot increasing its effective force through objects found in the environment.

Chapter 6 - Navigation Using Manipulable Objects with Only Onboard Sensing extends the NUMO domain to cases where the robot only has onboard sensing. A framework is presented that allows a robot to reach its goal by deciding *online* to use environment objects as tools.

Chapter 7 - Conclusion provides concluding remarks and outlines future research directions.

CHAPTER II

RELATED WORK

The research presented in this thesis is related to existing work in NAMO planning, planning under uncertainty, and robotic tool use. In addition, as the robot needs to perceive environment objects through onboard sensing, we also discuss related work in object detection. We now cover each topic in turn.

2.1 NAMO Planning

Navigation and manipulation planning poses a significant computational challenge even with *complete environment information*. Wilfong [107] first proved that deterministic NAMO with any number of obstacles is NP-hard. Demaine [19] further showed that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard.

In [92], Stilman presented a planner that solved a subclass of NAMO problems termed LP_1 where disconnected components of free-space could be connected independently by moving a single obstacle. The planner was able to solve the hard problems presented in [15] and was successfully implemented on the humanoid robot HRP-2 [93]. Subsequent work presented a probabilistically complete algorithm for NAMO domains [104]. However, all these methods solved NAMO assuming perfect knowledge of the environment and deterministic action outcomes.

Hauser presented the Minimum Constraint Removal problem (MCR) in [30] and showed its relation to the NAMO problem. The MCR problem aims to find the fewest geometric constraints that have to be removed in order to connect two configurations. NAMO can be seen as an instance of the MCR problem. However, the MCR formalism does not reason about how to remove constraints or about uncertainty in the process of removing constraints. In [56] we presented a planner that enables multiple robots to rearrange multiple objects. The planner can be used to deploy a multi-robot NAMO system. However, the planner again assumes perfect environment knowledge.

Wu [36] and Kakiuchi [41] introduced the first extensions to NAMO in *Unknown Environments*. In [36] a planner was presented that could solve NAMO problems given with substantial lack of initial state information. We extended this planner in [65] to return locally optimal solutions. However, both [36] and [65] are not capable of handling uncertainty, and instead assume that everything that comes within sensor range is given to the robot as ground truth. Further, actions are assumed to be deterministic. [41] presented a system that executes NAMO in unknown environments on the humanoid robot HRP-2 with only onboard sensing. However, the authors took a reactive behavior-based approach, rather than attempting full decision theoretic planning and as such the system is not able to bias its decision making towards plans that are likely to succeed.

2.2 Planning under Uncertainty

There are two major threads of work related to planning under uncertainty: research presented in the robotics or AI literature and in the general decision theoretical literature. We cover each in turn.

2.2.1 Robotics and AI

One class of algorithms specifically addresses mobile robot navigation with perfect sensing and actuation, but with fundamental initial uncertainty about the poses of obstacles in its workspace. These algorithms operate in a discretized state and action space, and employ methods based on A^* search. They assume that space is free unless it is known to be blocked, and replan when the current plan is made infeasible by newly discovered obstacles. Early versions [111] used a relatively straightforward planning strategy; more modern versions employ important algorithmic techniques to ensure computational efficiency and limit the frequency and scope of replanning. [48, 49, 91]. Likhachev *et al.* [66] integrate these methods with anytime algorithms, providing a replanning architecture that can deliver a feasible but sub-optimal plan quickly, but that converges to the optimal solution given more running time. In all of these methods, replanning is either triggered when the current plan is infeasible, or carried out as an ongoing computational process, at the highest rate possible, in response to new information gathered in real time. A successful urban autonomous vehicle [103] integrates dynamic replanning, which is triggered by plan invalidation due to a variety of environmental conditions. The work mentioned so far has concentrated on basic navigation, which is a relatively low-dimensional problem. For high-dimensional problems such as grasping, uncertainty is actively reduced by Dogar [22] by means of pre-grasp motions. The PPCP method [67], attempts to improve both efficiency and optimality by taking uncertainty into account when making initial plans.

Early work in symbolic planning for robots addressed issues of execution monitoring, replanning on failure, and exploiting serendipity on the Shakey [24] and Hilaire [88] robots. Recent related work in general symbolic planning solves problems with deterministic actions, but partial initial knowledge and partial sensing, by solving a sequence of classical planning problems [9, 10], with replanning triggered when the current plan becomes invalid. Similar methods can be applied in stochastic domains, through determinization [110]. An alternative approach is to delay decision-making until information is gathered, rather than attempting to create conditional or conformant plans in advance [23]. Fritz and McIlraith [26] consider symbolic planning in dynamic domains, including the ability to respond to changes that happen during planning.

Bratman [11] formulated a theory of rational action, which emphasizes the role of commitment to plans as a way of decreasing the burden of reasoning, and that addresses the problem reconsideration and the trade-offs it presents between efficiency and optimality. This theory was expanded into a computational architecture, IRMA [12, 33, 77].

2.2.2 Decision Theory

As discussed above, there are two basic forms of uncertainty that may affect a planner: uncertainty in the *action outcome*, and uncertainty in the world *state*. In the decision theory planning literature, these types of uncertainty are typically modelled in different ways. The first is captured by a *probabilistic action model*, and is the basis for the Markov Decision Process (MDP). The second requires augmenting the MDP with a *probabilistic observation model*, into the so-called Partially Observable MDPs (POMDP).

POMDPs have been applied separately to *navigation* planning by Koenig and Pineau [46] [75] and *grasping manipulation* by Hsiao [35]. While both domains are related to problem statement in this thesis, they focus on the configuration space of a single robot or a single object. In contrast, the domains discussed in this document require the robot to reason about the full set of objects in its workspace. Existing robot planners that use POMDPs are generally restricted to low-dimensional configuration spaces. This constraint holds even when applying the most recent approximate POMDP solvers such as point-based value iteration [75], belief compression [80] and Milestone Guided Sampling [51].

Several techniques have been developed for extending the MDP model to hierarchical tasks. Among the most well known include the options framework [99], hierarchies of abstract machines (HAM) [74], and the MAX-Q framework [20]. All three of these approaches rely on a generalization of the MDP to incorporate non-atomic, or *semi-markov* actions. The resulting model is referred to as a semi-Markov decision process, or SMDP [34].

The primary difference between these approaches is whether they involve *simplifying* or *augmenting* the original MDP. The goal of the options framework is to introduce abstract actions without compromising the finer-grained planning of the original MDP. Therefore options-planning does not offer any *representational* abstraction: all planning is done in the state space of the original MDP. In HAM, the system designer specifies a hierarchical collection of state machines for solving sub-tasks. This state machine presents an abstract interface for the high-level task, which can again be solved as an SMDP. The drawback of these approaches is that the resulting SMDPs are either too low-level for tractable task planning (options), or too high-level and modular for solving context-sensitive subtasks (HAM).

The third approach, MAX-Q, strikes a balance between Options and HAM. It is well-suited for the domains discussed in this thesis because it does not require predefined subtask policies, but still utilizes an abstract planning representation. We utilize the MAX-Q formalism for discretized NAMO environments. However, MAX-Q still assumes that each subtask is solvable using standard *dynamic programming* methods [20]. These algorithms are based on the theoretical results for the SMDP, which scales at best linearly in the size of the (non-abstract) state space [38]. This prohibits a direct application of the MAX-Q framework to the domains discussed in this thesis.

An alternative to dynamic programming for solving MDP is referred to as Sparse Sampling or Monte Carlo tree search (MCTS). The MCTS literature describes a family of algorithms which were introduced to provide MDP solvers which scale *independently of the size of state space* [43]. However, MCTS remains exponential in the depth of its search tree, which for the domains discussed in this thesis can often require tens or hundreds of primitive actions. Several heuristics have been developed for MCTS, including UCT [45] and FSSS [106], which are significantly more sample efficient than vanilla MCTS. While these algorithms are good candidates for a subtask planner, they cannot solve the overall problem of autonomously changing the environment topology even for discretized cases due to the large depth, branching factor, and sparsity of rewards in the domain.

For continuous environments, an alternative to MCTS is Monte Carlo simulation [54]. In the robotics literature Monte Carlo simulation techniques are widely used for localization and mapping [101]. We utilize Monte Carlo simulation to reason about action uncertainty within continuous NAMO domains.

2.3 Tool Use

Most existing forms of robotic tool use are focused on accurate positioning and control of specific tools such as welding instruments [17], spray guns [14], drills [21] and surgical instruments [3, 18]. In all of these scenarios the robot performs a well defined task with the tool. The autonomous discovery of functional properties of environment objects has recently gained interest. Stoytchev [98] and subsequently Sinapov [87] suggested the investigation of arbitrary objects through interaction and connecting their shape to a particular function. While the control methods developed for these scenarios are complementary to the systems discussed in this thesis, we do not assume that the robot is given a specific task to accomplish with a specific tool or is tasked with determining all possible affordances [28] of an object. Instead, this document considers scenarios in which a robot has to autonomously determine if and how it needs to use environment objects as tools.

An extensive survey on early AI research on tool use can be found in [7].

2.4 Object Detection

The detection of objects in 3D laser data has been studied intensively in various research fields. As such, we are only addressing the work most relevant to partial object and furniture detection in human environments.

In [83] Rusu et al. present a system for the acquisition of hybrid Semantic 3D Object Maps for indoor household environments based on 3D point cloud data. The authors use a two step approach for detecting kitchen furniture based on the detection of horizontal and vertical planes as well as knobs.

Blodow et al. describe a mapping system acquiring 3D object models for indoor environments in [8]. The authors present a system for segmenting and geometrically reconstructing cabinets, tables, drawers and shelves based on multiple scans of the objects. Tables, the most relevant part to our work, are detected by finding horizontal surfaces within a given height range, which this does not allow to distinguish tables and shelves. Holz et al.

are using 3D Time-of-Flight cameras in [32] for semantic scene analysis. The authors are using an MSAC-based approach and surface information in a point's local neighborhood to detect table tops. This is done by assuming that a table top point has a surface normal nearly parallel to the z-axis and that the local surface is smooth. MSAC is used to fit planar surface models into the table point set. This approach yields the same limitations as [8].

Johnson et al. presents in [37] a shape-based object recognition system based on matching Spin Images. Spin Images however require a high resolution image, and as such are difficult to use in 3D point clouds obtained by a laser.

Marton et al. incorporate in [71] 3D laser scans and 2D vision data for object classification. They use the Radius-based Surface Descriptor (RSD) on 3D data, extract the region of interest into a camera image, and compute a 2D SURF vector for each patch. The final classification is performed through a support vector machine. However, it remains unclear how this work could be extended to perform well with objects typically encountered in NAMO or NUMO domains, such as furniture, as these objects typically lack texture. In addition, partial occlusions would be difficult to handle by this approach.

Steder et al. [89] demonstrated the detection of chairs and other objects in 3D point clouds using point features from range images. The authors use euclidean distance in a vector space spanned by Harris feature vectors to find candidates. GOODSAC is used to find a model transformation and false positives are rejected based on a scoring function using scaled range images. Steder et al. also presented the normal aligned radial features (NARF) in [90]. Interest points are detected on stable surface areas with significant local changes. A descriptor is then computed by overlaying a star pattern on the range image generated by looking at the interest point along the estimated normal for this point. The authors also describe the potential of matching the feature descriptors as an object recognition approach. However, we experimented with NARF and found that the descriptor is only of limited use in object recognition due to a lack of local texture in range images and the loss of valuable orientation information during normal alignment, *e.g.* a horizontal surface becomes indistinguishable from a vertical one.

Mozos et al. [72] demonstrate a method of categorizing partially occluded objects from object parts learned from segmented 3D models, which are first segmented based on the object's structure. The database of segmented parts is used to suggest categorizations from a scene. The candidates are combined through Hough voting and verified through model fitting. As this approach segments based on the object's structure, the segmented parts do not correspond exactly with occlusions caused in real scenes, which is a property of the occluding object and the scene, not the occluded object. It remains unclear whether [72] is sensitive to partial occlusion of the segmented parts.

Viewpoint Feature Histograms as presented by Rusu et al. [82] are encoding geometry as well as viewpoint information into a descriptor. The authors demonstrate the effectiveness of the descriptor on a dataset consisting of more than 60 unoccluded kitchenware objects. We utilize Viewpoint Feature Histograms in this thesis and summarize the core idea in Chapter 4.1.1.

CHAPTER III

NAMO WITH ACTION UNCERTAINTY

In this chapter we present the first decision theoretic planning framework for the Navigation Among Movable Obstacles (NAMO) domain [92, 93, 104]. Future rescue robots tasked with saving humans from disasters such as floods and earthquakes will be presented with challenging manipulation decisions. In the face of displaced furniture and rubble, traditional motion planning algorithms searching for a collision-free path from start to the goal are not sufficient if all paths to the goal are blocked by obstacles. However, even given a map of the environment, how does the robot decide which path to take, or which objects to move?

As discussed in Section 1.2, there are two primary challenges in developing a practical algorithm for such situations: the dimensionality of the state space and the inevitable uncertainty in action outcome when interacting with physical systems. Prior work on NAMO focused largely on handling the problem of *dimensionality*, and has not yet addressed the underlying issue of *uncertainty*. In reality, robots have noisy actuators and sensors as well as incomplete world models.

Leveraging ideas from decision theory, we have achieved a novel representation that formally addresses action uncertainty in NAMO for scenarios that require the robot to move a single object out of the way to connect two free-space regions. By casting the NAMO problem as an abstract Markov Decision Process (MDP), we define the NAMO MDP. In this chapter we present different methods for computing and solving the NAMO MDP, and as such demonstrate the first NAMO planners that bias their decisions at plan time in order to compute policies that are likely to succeed[1]. The following system considers I_1UL

[1]This work appears in [59, 60].

problems (see Section 1.2).

3.1 Overview

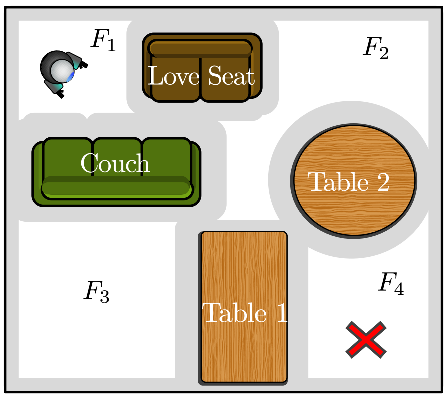
This chapter focuses on action uncertainty and assumes full knowledge of the position of the objects. The exact object properties, however, are assumed to be uncertain.

Our general strategy to handling action uncertainty in NAMO is to construct an MDP that closely resembles the real problem but can be solved in linear time for typical environments. The construction of this MDP builds on two insights of the domain. First, there is a natural abstraction from the low-level state space into a small set of abstract states, which we call *free-space regions* to indicate that the robot can move collision-free between any two configurations within the region. Second, there are a small number of implied abstract actions for maneuvering in this abstract state space: since each free-space region is circumscribed by obstacles, we can define the abstract action "create an opening to neighboring free-space" for each obstacle. Together these two ideas permit the construction of an abstract MDP. Fig. 3 visualizes an example.

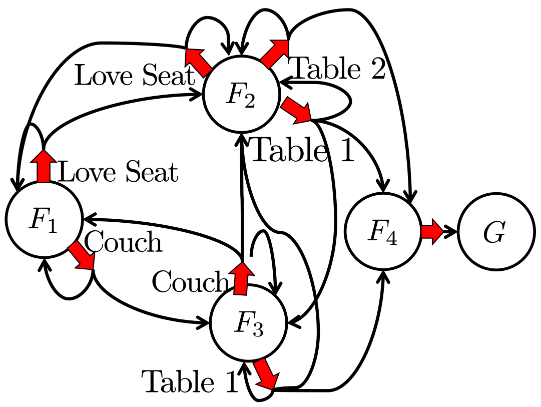
The abstract MDP representation by itself, however, is insufficient for creating tractable planners: to solve the MDP, the abstract actions have to be grounded and evaluated in the underlying non-abstract state and action spaces. Even fully solving these manipulation subtasks alone is intractable. Our general approach to overcome this challenge will be to approximate the transitions and rewards in the abstract MDP using Monte Carlo methods.

3.1.1 Discretized Environments

For discretized environments we embed the NAMO MDP in the MAX-Q formalism (Section 3.3.1.1) and incorporate a complementary approach called Monte Carlo Tree Search (MCTS, Section 3.3.1.3). The MAX-Q formalism enables us to construct the abstract NAMO MDP in a theoretically well defined way, while MCTS allows us to obtain estimates for the relevant parts of the subtask policies. MCTS is a technique for generating a policy for one state at a time, with runtime that depends on the length of the plan rather



(a) free-space components



(b) Resulting high-level MDP

Figure 3: Concept visualization. The presented framework defines an abstract MDP operating with abstract states representing free spaces and abstract actions reflecting the notion of “creating an opening” by manipulating a specific obstacle. Consider the left most state of the MDP in (b). If the robot is in this state, physically in free space 1, it has the possible actions of manipulating the Love Seat or the Couch. This may grant the robot access to free space 2 or 3, respectively, or fail, leaving the robot in free space 1.

than the number of states. MCTS is well-suited for the individual manipulation tasks for two reasons. First of all, the NAMO MDP only requires a manipulation policy from a few starting locations of the obstacle, which are known at plan-time. Second, since the abstraction divides the problem into smaller manipulation tasks, the overall policy length for any subtask is quite small. These two properties allow us to substitute MCTS in place of value-iteration in the MAX-Q formalism of the NAMO MDP, without compromising the hierarchical optimality for the overall algorithm.

3.1.2 Continuous Environments

While the solution approach for discretized environments yields insight into the domain, it is not always possible to discretize the continuous configuration and action spaces of a realistic robotic system without sacrificing either runtime or resolution. We therefore present a second method for obtaining and solving the NAMO MDP in Section 3.4 applicable to continuous environments. The approach builds on a combination of modern sample-based planners and Monte Carlo simulation to obtain estimates of the transition and reward function for the NAMO MDP. These estimates enable us to use standard dynamic programming techniques to solve the NAMO MDP directly.

3.2 NAMO MDP

In this section we formally introduce MDP and define the NAMO MDP, which closely resembles the real problem, but can be solved in realtime. Section 3.3 and Section 3.4 outline how to construct and solve it for the discretized and continuous environment case, respectively.

3.2.1 Preliminaries

3.2.1.1 Markov Decision Processes

The Markov Decision Process (MDP) is a model for stochastic planning, and the foundation of the presented framework. We define an MDP $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$ for a finite set of states S, a finite set of actions A, a transition model $T^a_{ss'} = P(s'|s,a)$ specifying the probability of reaching state s' by taking action a in state s, a reward model $R^a_s = r(s,a)$ specifying the immediate reward received when taking action a in state s, and the discount factor $0 \leq \gamma < 1$.

The standard technique for solving MDPs is *Value Iteration* (VI) [81], which is used to find the optimal policy $\pi^* : f(s) \to a$ which maps states to actions in order to maximize the expected long-term reward, or value $V(s)$ for all states $s \in S$. This value function is defined according to the Bellman recursion:

$$\begin{gathered} V(s) = \max_a \left[Q(s,a) \right] = \\ \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right] \end{gathered} \tag{1}$$

VI is a dynamic programming solution to solving Eq. 1 which is polynomial in the number of states (with fixed error ε).

3.2.2 NAMO MDP Definition

The proposed NAMO MDP is an abstraction of the NAMO domain, with navigation between free-space regions as the abstract task. Recall that an MDP is defined as $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$, which leaves four properties to define (γ is a parameter of the problem specification).

States and Actions: The fundamental state space in the NAMO problem is the set of possible configurations $\mathscr{C}_W$ of the robot and environment. We define the high level NAMO MDP M_h as an abstraction of this low-level state space. The state space S_h is defined to be the set of free-space regions implied by the starting obstacle configuration. The action space A_h is the union of all possible manipulation sub-tasks for each region, where sub-task $a_h^{ijk} = \pi_l^{ijk}$ means "open a path from state i to state j by manipulating obstacle k". The set of possible obstacles k for each starting state i are defined to be the bounding obstacles for the corresponding free-space region.

Transitions and Rewards: R_h is defined according to the expected cost for creating a specific opening in the underlying non-abstract state space. For example, in Fig. 2(a) the policy π_l^{ijk} might be to move from $s_h = F_1$ to F_2 by moving the table. The reward associated with this policy then comes from the sum of discounted rewards obtained in the *low-level* state-space while executing π_l^{ijk}. Section 3.4.3 considers various ways to define low-level costs to obtain favorable behavior in natural domains.

The transition probabilities in M_h directly represent the expected outcome of executing a subtask π_l^{ijk} in some state s_h. The manipulation sub-task π_l^{ijk} terminates in state j if it successfully opens a path from i to j, and terminates in state i otherwise[2]. Therefore, the transition probability $P(s' = j | s = i, \pi_l^{ijk})$ is positive if and only if the free-spaces represented by s_i and s_j are separated by the obstacle k. This suggests that transition model T_h is sparse: the probabilities are automatically zero for all states that do not share an obstacle.

[2] We do not consider unintended secondary openings.

For example, in Fig. 6(a), the transition probability for $P(s' = F_6 | s = F_1, \pi_{F_1 F_6 O_3})$ depends on whether the robot can move O_3, a green couch, out of the way. Section 3.4.3 discusses how to parameterize obstacles in a way that reflects their true manipulation dynamics.

Note that this construction is an instance of what Dietterich refers to as a funnel abstraction [20]: the value of all possible robot configurations (positions) within the target free-space get mapped to a single value: the value of that region. This is the basic abstraction from which the NAMO MDP obtains its savings. However, to solve the NAMO MDP, the sub-task policies π_l^{ijk} have to be solved. Given the exponential size of the low-level state space, this is prohibitively expensive to do exactly. The formalism is therefore, by itself, not sufficient.

In the following section we outline the use of a complementary method for the discrete case, provide a concrete implementation, and argue its usefulness. In Section 3.4 we analyze the continuous case, and provide an analogous solution. The following sections will assume a 2D projection of the environment, however this is not an intrinsic restriction of the NAMO MDP.

3.3 Discrete Environments

For a discretized environment, we embed the NAMO MDP in the MAX-Q framework and make use of MCTS to obtain estimates of the subtask values. While substantially different, MAX-Q and MCTS are both techniques developed for handling large state spaces in an MDP. To our knowledge, these techniques have never been combined. Consequently they will first be introduced independently, with the remainder of the section demonstrating how they can be efficiently combined to solve the NAMO MDP.

3.3.1 Preliminaries

3.3.1.1 MAX-Q Value Function Decomposition

The MAX-Q framework is a technique within the reinforcement learning literature which describes how a *value function* for an MDP may be decomposed according to a *task hierarchy*. To understand this idea, consider a two-level hierarchy composed of a high-level policy π_0, defined over a set of subtask policies $\pi_i, i \in \{1, n\}$, which are in turn defined over primitive actions. That is, $\pi_0 : f(s) \to \pi_i$, and $\pi_i : f(s) \to a$.

The key insight behind MAX-Q is that the value functions for the subtask policies $V_{\pi_i}(s)$ contain all the information needed to represent the value function of the parent policy $V_{\pi_0}(s)$. This is true because, according to the Bellman equation for SMDPs, the value of executing subtask π_i in state s is simply the sum of the (discounted) reward accumulated by π_i itself, and the future expected reward of whichever state π_i terminates in:

$$Q(s, \pi_i) = R(s, \pi_i) + \sum_{s', \tau} P(s', \tau | s, \pi_i) \gamma^{\tau} V(s') \tag{2}$$

where γ^{τ} ensures that the value of s' is appropriately discounted according to the time τ that π_i took to terminate [5].

The first term in this expression, $R(s, \pi_i)$, is precisely the information encoded by $V_{\pi_i}(s)$:

$$R(s, \pi_i) = V_{\pi_i}(s) = \mathbb{E}_{\pi_i}\left[\sum_{t=1}^{\tau} \gamma^t R(s, a)\right] \tag{3}$$

Therefore, planning in the high-level task simply involves using the values of the subtasks as immediate rewards for their execution in the high-level policy.

In addition to an analysis of compositional value functions, the full MAX-Q framework also describes several model-free learning algorithms. However, we are instead interested in model-based planning, in order to take advantage of the robot's knowledge of the effects of its actions. Our formalism therefore differs in some important details, but shares the primary data structure (a hierarchical Q-function) and general format of "MAX-QQ" [20].

3.3.1.2 Types of optimality

So far, we have described a *bottom-up* representation of value functions, in which values of subtasks are *projected up* to parent tasks. This approach provides space efficiency, as well as the opportunity to divide and conquer: each subtask can be learned separately and combined to learn the parent task. Dietterich [20] refers to the class of policies which can be represented by such a model as *recursively optimal*, meaning all policies are optimal given optimal solutions to their subtasks.

The drawback to this approach, which makes it inappropriate for the NAMO domain, is that subtasks are learned without knowing their context in the overall plan. This is a problem, for example, if moving two obstacles provides the same reward, but only one of them opens a path to the goal. In general, tasks with sparse rewards tend to require either additional *shaping rewards* for the subtasks, or a solution strategy that includes *top-down* information [5].

We take the top-down approach by using the value of the target free-space region as a reward for the manipulation policy that clears the appropriate obstacle. This is equivalent to Dietterich's solution for the model-free case, in which the completion functions (the right-most term of Eq. 2) are treated as subtask terminal state rewards [20]. Policies that are learned in this fashion are referred to as *hierarchically optimal*, meaning that the overall policy is optimal given the constraints of the imposed hierarchy [20].

3.3.1.3 Monte Carlo Tree Search

MCTS was developed as a way of allowing near-optimal planning for MDPs with large or infinite state spaces. The main idea is to relax the goal of computing a full policy, and instead focus on computing the optimal policy for a single state – the state the agent is in. In their original work on *sparse sampling*, Kearns et al. showed that it was possible to obtain ε-optimal Q-value estimates for the current state from a set of sampled transitions, and that the number of samples C per state was independent of $|S|$ [43].

MCTS works by building a search tree from the current state, selecting actions according to some search policy π_s, and sampling transitions from the transition model $T^a_{ss'}$ for each action. This tree generates a set of sampled rewards, which can be backed up to the root node to obtain a Q estimate according to:

$$Q^d_{SS'}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{d-1}_{SS'}(s',a') \tag{4}$$

$Q^d_{SS'}(s,a)$ refers to the expected value of taking action a in state s, and following the optimal policy for $d-1$ subsequent steps (so $Q^1_{SS'}(s,a) = R(s,a)$). Equation 4 defines a simple recursion for using forward simulation to select actions, despite uncertainty in the action model.

3.3.1.4 The effective horizon

There is one additional result from the MCTS literature, regarding search depth, which we exploit in our hierarchical NAMO framework. Kearns et al. proved in [43] that MCTS could achieve ε-optimality with an $O((|A|C)^H)$ running time, where H is the *effective horizon* of the problem. Based on the fact that rewards in the distant future have little effect on the Q-values of the current state, this proof bounds H according to ε and R_{max} (the maximum possible reward):

$$H = \left\lceil log_\gamma \left(\frac{\varepsilon(1-\gamma)^2}{4V_{max}} \right) \right\rceil, V_{max} = R_{max}/(1-\gamma) \tag{5}$$

Equation 5 states that the effective horizon *increases* with the value of the maximum possible reward that can be achieved. As shown below, this relation can be exploited in a hierarchical setting to have the MCTS manipulation planners spend time in proportion to the value of their target free-space region.

Note that MCTS alone, however, is inappropriate for the NAMO domain. Its applicability is limited by the branching factor, task horizon, and sparsity of rewards in the domain. We now embed the NAMO MDP in the MAX-Q framework, demonstrate how the NAMO

MDP can be constructed for discretized environments and show how it can be solved using MCTS.

3.3.2 Discrete NAMO MDP Construction

The MAX-Q formalism provides a theoretically well defined technique to ground the abstract NAMO MDP in the underlying non-abstract state space for discretized environments. To achieve this, we introduce an additional lower-level MDP M_l.

M_l operates on the raw configuration space, and is used to model the manipulation actions of M_h. The state space S_l is the set of possible configurations of the robot and environment and the action set A_l is the set of primitive action for manipulating obstacles on the map. Similarly, the transition model T_l encodes the domain uncertainty in terms of action primitive execution uncertainty. The reward function of the NAMO MDP and of M_l are now defined in accordance to their hierarchical relationship. Recall that the reward function R_h reflects the reward of executing an action $a_h^{ijk} = \pi_l^{ijk}$ in the abstract NAMO MDP. The MAX-Q formalism enables us to now ground R_h by setting it to the expected reward for executing subtask π_l^{ijk} in M_l. The reward function for M_l in turn sets the terminal reward for executing π_l^{ijk} as the value of its final state in M_h. Note that this is the context-sensitivity requirement described in Section 3.3.1.2. This should be intuitive, because the high-level policy needs to see the actual outcome of executing each possible subtask, and each subtask needs to see its *context* in the high-level policy in order to know how important different obstacles are. We now provide details for obtaining the NAMO MDP before we show how it can be solved by using MCTS to obtain estimates for the subtasks.

3.3.2.1 State and Action Space

In order to construct the NAMO MDP, the system needs only to determine the high level state space S_h and action space A_h, as the low level state space S_l and action space A_l are given by the domain specification[3]. To compute S_h and A_h we need to find the free-space regions and disconnecting obstacles. In our implementation we identify the free-space regions by performing seeded wavefront expansions on $\mathscr{C}_W$ to determine disjoint regions. These disjoint regions yield S_h. During this operation we also save the list of obstacles bounding each free-space region into an adjacency list $\mathscr{L}_r$. These represent the possible manipulation targets for that state, the union of which yields A_h.

We now detail on how we encode the domain uncertainty in our implementation.

3.3.2.2 Model Representation

The low level transition model T_l encodes the uncertainty in the NAMO domain that arises from manipulation actions. This implies a *displacement model* for each object, which depends on the action a being executed, and the target obstacle o:

$$\delta x, \delta y \sim P(\delta x, \delta y | a, o) \tag{6}$$

3.3.3 Solving the NAMO MDP

Section 3.3.2 showed how to use MAX-Q to ground the NAMO MDP as a two-level hierarchy with components defined in terms of each other: the rewards for M_l were obtained from the values of states in M_h, and the values of states in M_h were defined based on outcomes of subtasks in M_l. With this formulation, values in both M_l and M_h reflect the true transition probabilities and rewards for manipulation actions.

This suggests an iterative scheme in which we alternate between updates to the high-level policy π_0 given the current values for subtasks $V_{\pi_l^{ijk}}$, and updates to the individual subtasks π_l^{ijk} given the values in V_{π_0}. However, computing these values requires actually solving the associated MDPs, which was shown to be intractable for M_l in Section 1.2, since $S_l = \mathscr{C}_W$, the set of possible configurations of the robot and environment.

[3]For simplicity, we used axis-aligned manipulations as action primitives in our system, but in practice these would typically be replaced with a more appropriate choice for physical systems, such as those introduced in [92].

Fortunately, sparse-sampling planners provide a way of obtaining approximate solutions to M_l, and make assumptions which are compatible with our hierarchical approach for discretized environments (Section 3.3.1.3). Therefore, our actual algorithm performs the same alternating policy-iteration scheme described above, but with the substitution of an MCTS planner in place of value-iteration for the manipulation MDPs. These MCTS planners compute Q-values in $\mathscr{C}_W \times A_l$, and sample transitions from the displacement model defined in Eq. 6. Sampling terminates when an opening is achieved or the maximum search depth, defined by the horizon H (Eq. 5), is reached. For a general MCTS planner, H is defined as a function of R_{max}. In combination with a hierarchical policy iteration, however, it can be used to explicitly force the computation effort for each subtask planner to scale in proportion to its estimated utility. This is achieved using a *dynamic horizon*. Dynamic horizon recomputes the H-value, Eq. 5, of each MCTS planner *separately* based on the value of the target state in S_h. The overall effect of this operation is that π_i does not waste time sampling actions past the point where rewards can significantly affect the Q-value of the subtask. Algorithm 1 provides pseudocode for constructing and solving the NAMO MDP in discretized environments.

The following section will argue the usefulness of this complete framework.

3.3.4 Evaluation

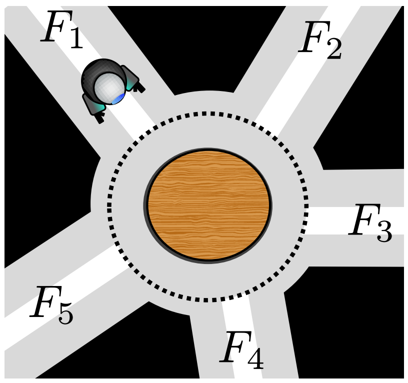
This section provides a theoretical proof and empirical data that our algorithm for solving the NAMO MDP in discretized environments has a run-time which is linear in the number of obstacles, for non-degenerate environments.

3.3.4.1 Theoretical Analysis

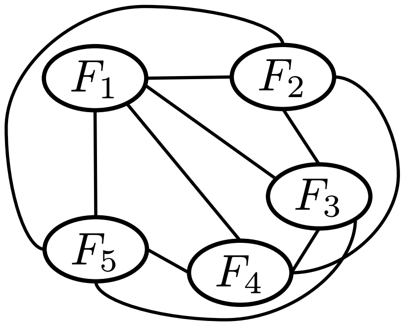
The following analysis is based on relating the sparsity of the free-space transition matrix T_h to the *adjacency graph* G_A over free-space regions. The sparsity of T_h directly controls the complexity of value iteration.

Algorithm 1: Proposed framework for discretized environments.

```
Input: $O$: obstacles, $P(\delta x, \delta y | a_l^1, o_1) \dots P(\delta x, \delta y | a_l^p, o_m)$: displacement models,
        $\mathscr{C}_{goal}$: goal configuration
Output: $\pi_h$: high level policy, $\Pi_l$: dictionary of low level policies
1  $F \leftarrow$ GET_FREESPACES($O$);// wavefront
2  $M_h(S_h, A_h, T_h, R_h) \leftarrow (F, \{\}, \mathbf{[0]}, \mathbf{[0]})$;
3  $V \leftarrow \emptyset$; $\pi_h \leftarrow \emptyset$; $\Pi_l \leftarrow \emptyset$;
   // determine high level MDP definition:
4  foreach $s_i \in S_h$ do
5      if $s_i$ contains $\mathscr{C}_{goal}$ then
6          $\forall a\ R[(s_i, a)] \leftarrow$ utility of reaching the goal;
7      foreach $o_k$ adjacent to $s_i$ do
8          foreach $s_j \in S_h$ do
9              if $o_k$ adjacent to $s_j$ then
10                 $A_h \leftarrow A_h \cup \{a_h^{ijk}\}$ ;
11                 $T_h[s_h^i, a_h^{ijk}, s_h^j] \leftarrow 0.5$;
   // run value iteration:
12 while error not within $\varepsilon$ do
13     foreach $s_i \in S_h$ do
14         $v \leftarrow 0$;
15         foreach $a_h^{ijk} \in A_h$ do
16             $h \leftarrow \lceil log_\gamma(\frac{(\varepsilon(1-\gamma)^2)/4}{s_h^j.v}\rceil)$;
17             $(q_k, \pi_l^{ijk}) \leftarrow$MCTS($a_h^{ijk}, h,$        $P(\delta x, \delta y | a_l^1, o_k), \dots, P(\delta x, \delta y | a_l^p, o_k)$);
18             if $q_k > v$ then
19                 $v \leftarrow q_k$;
20                 $\pi_h(s_h^i) \leftarrow a_h^{ijk}$; $\Pi_l(o_k) \leftarrow \pi_l^{ijk}$;
21         $V(s_h^i) \leftarrow v$;
22 return $\pi_h, \Pi_l$;
```



(a) Environment



(b) Resulting adjacency graph G_A.

Figure 4: Example of a non-planar G_A.

Definition 1. *The adjacency graph $G_A = (V,E)$ is defined to have vertices $v_i \in V$ which uniquely represent free-space regions F_i, and edges $e(i,j) \in E$ connecting adjacent free-space regions. That is, $e(i,j) \in E \Leftrightarrow adjacent(F_i,F_j)$, with $adjacent(F_i,F_j) = true$ iff F_i and F_j are disconnected through a single obstacle.*

Figures 4 and 6 show examples of workspaces and their associated adjacency graphs. A graph G is planar if it can be drawn on the plane with no edges crossing except at common vertices. Kurarowski's theorem states that a graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$, where K_5 denotes a complete graph with five vertices, and $K_{3,3}$ denotes a complete bipartite graph on six vertices [100]. Note that G_A in typical environments will fulfill this definition. Contrary examples include:

(a) A set of five free-spaces that are all adjacent to each other, separated by only a single obstacle.

(b) A set of three free-spaces that are all adjacent to a disjoint set of three free-spaces, but not adjacent to each other.

Fig. 4 shows one of these degenerate cases.

Further, recall that by definition of T_h, the only next-states with non-zero transition probability from state s_h are states representing free-spaces adjacent to s_h or s_h itself (failure case). This gives rise to the following lemma:

Lemma 1. *$T_h(s,\cdot,\cdot)$ has on average a constant number of next-states s' with non-zero probability if G_A is simple planar.*

Proof. First, it is trivially true that on average, $T_h(s,\cdot,\cdot)$ contains a constant number of self-transitions: $\frac{|S_h|}{|S_h|} = 1$. The remaining non-zero entries in T_h are a direct mapping of $G_A = (V,E)$. This suggests that the average number of next-states with non-zero probability in T_h is equal to the *average* degree of G_A. Recall that in a planar-graph there exists a constraint relating the number of edges e to the number of vertices v (see Appendix A):

$$e \leq 3v - 6 \tag{7}$$

Eq. 7 directly implies a bound on the average degree d_{avg} of G_A. Let d_i denote the degree of vertex v_i, then:

$$d_{avg} = \frac{\sum_{i=1}^{v} d_i}{v} = \frac{2e}{v} \leq \frac{2(3v-6)}{v} = 6 - \frac{12}{v} < 6 \tag{8}$$

Consequently, $T_h(s,\cdot,\cdot)$ has on average at most 6 next-states with non-zero probability. Importantly this bound is over all possible actions for a given state (recall that there are only two possible outcomes for *single* action). If m is the number of actions available in s_h and l is the number of possible outcomes per action, then we have $ml \leq 6$. Finally, the expected total number of non-zero entries $k_{T_h^+}$ is:

$$\begin{aligned} k_{T_h^+} &= E\left[\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{n} P\left(T_h(s_i,a_j,s_k) > 0\right)\right] \\ &= \sum_{i=1}^{n} d_{avg} \\ &\leq 6n \end{aligned}$$

□

Lemma 2. *The run-time of the proposed framework for discretized environments is linear in the number of obstacles $|O|$ if the adjacency graph G_A is simple planar.*

Proof. First, consider M_h. The number of states $n = |S_h|$ is linear in $|O|$. Value iteration is performed over M_h. Since the value-error ε is a free parameter, the complexity of value iteration is typically separated into the cost-per-iteration and the expected number of iterations.

I. The number of iterations required by VI to achieve an ε-error bound is

$$m = \left\lceil log\left(\frac{2R_{max}}{\varepsilon(1-\gamma)}\right)/log(1-\gamma)\right\rceil \tag{9}$$

which is independent of n [81].

II. In the worst case, the inner loop of value iteration is quadratic in n, due to the need to sum over full rows of T_h while computing the expectation over next-states in each Bellman update [38].

To see II, consider the inner loop update of the standard VI implementation:

$$\forall s : V(s) \leftarrow \max_a R(s,a) + \gamma \sum_{s' \in S} T_h(s,a,s')V(s') \tag{10}$$

However, since T_h is sparse for simple planar G_A (Lemma 1), VI can ignore all but the positive entries:

$$\forall s : V(s) \leftarrow \max_{a \in A^+} R(s,a) + \gamma \sum_{s' \in S^+} T_h(s,a,s')V(s') \tag{11}$$

where A^+ denotes the available actions for state s and S^+ denotes the states with positive transition probability $S^+ : T_h(s,a,\cdot) > 0$. Following Lemma 1 for simple planar G_A, we have that $|A^+||S^+|$ is on average constant. Thus the expected per-iteration cost of value iteration thus reduces to $6nc_a$, where c_a represents the expected cost due to action evaluations in the base MDP, considered next.

Recall that in M_h, the actions a_h actually represent manipulation policies π_l in a lower-level MDP defined over the raw configuration space of the environment. According to the MAX-Q hierarchical value decomposition described in Section 3.3.1.1, the Bellman update

Eq. 10 expands to:

$$\begin{aligned} V(s_h) &= \max_{\pi_l} R(s_h, \pi_l) + \gamma \sum_{s'_h} T_h(s_h, \pi_l, s'_h) V(s'_h) \\ &= \max_{\pi_l} V_{\pi_l}(s_h) + \sum_{s'_h, \tau} \gamma^{\tau} P(s'_h, \tau | s_h, \pi_l) V(s'_h) \end{aligned}$$

where subscripts h and l are included to make explicit the level in the hierarchy at which each quantity is defined, and $\pi_l \in A(s_h)$ denotes the set of manipulation actions available in high-level state s_h.

Next we require solutions for the quantities depending simultaneously on both s_l and s_h, namely $V_{\pi_l}(s_h)$ and $P(s'_h, \tau | s_h, \pi_l)$, that are independent of n. As shown in Section 3.3.1.3, we can approximate the value of a *specific* low-level state s_l using MCTS, yielding a value $V_{\pi_l}(s_l)$. Given an initiation set $\mathscr{I}_{\pi_l} \subseteq S_l$ for each subtask policy, we can obtain values for executing subtask policy π_l in the abstract state s_h by averaging:

$$V_{\pi_l}(s_h) \approx \frac{1}{k} \sum_{i=1}^{k} V_{\pi_l}(s_l \sim I_{\pi_l}) \tag{12}$$

We now make use of the NAMO funnel abstraction (Section 3.2.2) to obtain the high-level transition probabilities $P(s'_h, \tau | s_h, \pi_l)$ for a low-level policy π_l. Recall that for a particular subtask policy π_l the only two outcome possibilities in S_h are that the robot reaches the target state s^t_h or remains in the current state s^c_h. That is, $P(s'_h, \tau | s_h, \pi_l) = 0 \quad \forall s'_h \notin \{s^t_h, s^c_h\}$. Next recall from Section 3.3.3 that the horizon H_l for each subtask policy can be bounded according to the current value of the target free-space region $V(s^t_h)$, which guarantees that MCTS terminates. Let s^*_l be a terminal state for policy π_l obtained from the terminal nodes of an MCTS search tree evaluated to a depth H_l. Using the mapping $\phi(s_l) \to s_h$ defined by the funnel-abstraction, transition probabilities in S_h can be approximated by averaging over the terminal states in the MCTS tree:

$$P(s'_h, \tau | s_h, \pi_l) \approx \frac{1}{|S^*_l|} \sum_{s^*_l \in S^*_l} I(\phi(s^*_l) = s^t_h) \tag{13}$$

Together Eq. 12 and Eq. 13 provide approximate solutions to the quantities required for performing value iteration in the high-level MDP (Eq. 12). Furthermore, both are obtained from an MCTS tree evaluated at a finite set of initial configurations, which are independent of the total number of obstacles. Thus c_a is a constant, depending on R_{max}, γ, and the number of roll-outs k, but not the number of obstacles n. This yields an overall complexity of $O(n) = O(|O|)$ □

Convergence The standard convergence proof for VI assumes that the reward and transition distributions for the target MDP are stationary [81]. However, the complexity result in Lemma 2 required an approximation of the reward and transition probabilities for the high-level state space based on estimated outcomes in the low-level state-space. Thus we must also show that Eq. 12 and Eq. 13 are consistent estimators for their respective distributions R_h and T_h. Both estimators are ordinary Monte Carlo methods and are consistent by the law of large numbers. First consider the estimator for R_h:

$$\hat{R}_h = V_{\pi_l}(s_h) = \frac{1}{k}\sum_{i=1}^{k} V_{\pi_l}(s_l \sim I_{\pi_l}) \rightarrow N(R_h, \frac{1}{k}Var[R_h]) \tag{14}$$

As long as $V_{\pi_l}(s_l)$ provides *i.i.d.* samples of the true reward function R_h, then $\hat{R}_h \rightarrow R_h$ as $k \rightarrow \infty$. By construction, R_h is defined simply as the reward accumulated during a subtask execution, which implies that $V_{\pi_l}(s_l) \overset{iid}{\sim} R_h$. A similar argument applies to $\hat{T}_h$, which is also grounded in S_l and fully defines the transition dynamics of the abstract MDP, and therefore also provides *i.i.d* samples of T_h. Consequently both provide consistent estimators and VI converges.

3.3.4.2 Empirical Analysis

To evaluate the proposed framework for discretized environments, we have implemented it in simulation.

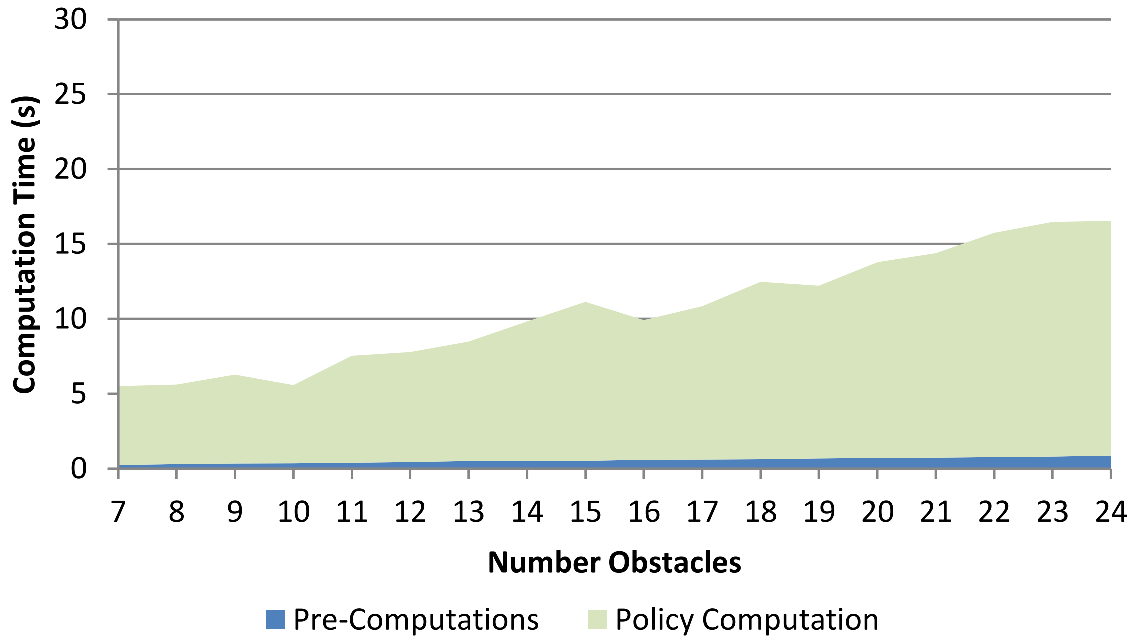
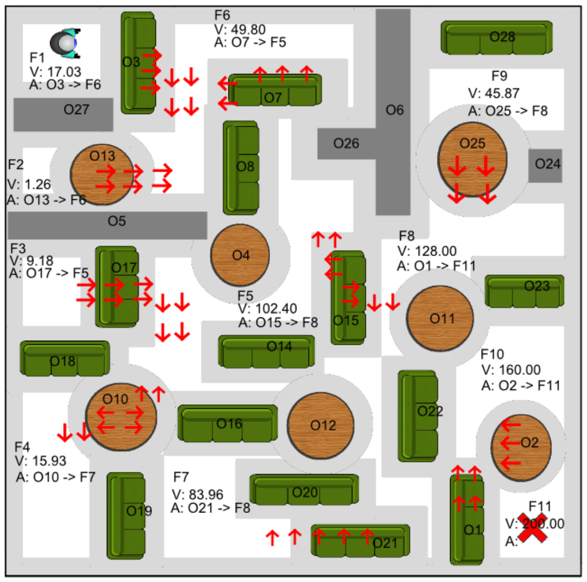


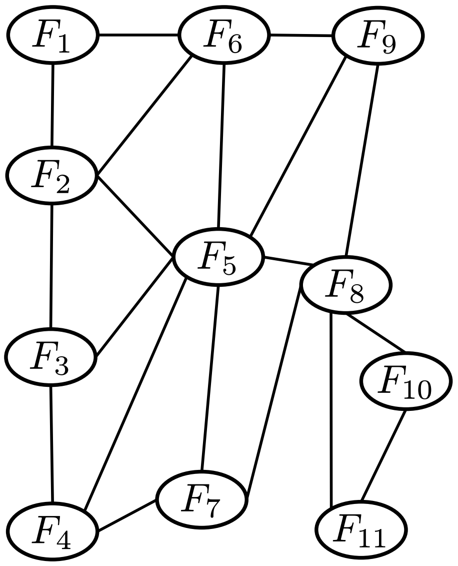
Figure 5: Obtained average computation times as a function of the number of obstacles. The graph suggest linear complexity in the number of obstacles.

Runtime We ran the framework on 1000 randomly generated NAMO environments. The size of the map was sampled uniformly to be between 250x250 and 400x400 grid cells. The number of obstacles for each map was uniformly sampled to be between 7 and 24, each obstacle in turn having random position, shape (rectangular or ellipsoid) and size (minimum 15x15 cells, maximum 65x65 cells occupation). Motion models $P(\delta x, \delta y | a_l, c)$ were represented using histogram discretized Gaussians with their mean and standard deviation randomly varying for different objects. The generated maps had an average of more than 70% cells occupied.

Fig. 5 summarizes the computation time as a function of the number of obstacles. Pre-computations include the generation of the configuration space representation and determination of free-space regions. While we show computation time as a function of number of obstacles, note that there are many other contributing factors, such as the particular configuration and number of free-spaces, the complexity of planning to create openings etc. Fig. 5 averages over these factors, and consequently resulted in a high standard deviation



(a) Obtained solution. The low level policies are visualized as a sparse vector field indicating the locally dominant directions.



(b) Adjacency graph.

Figure 6: Example environment. V is the states value and A denotes the action to execute. E.g. the free-space containing the robot has a value of 17.03 and the best action to execute is to manipulate obstacle $O3$ to gain access to free-space $F6$. Low level policies are visualized as a vector field. Static obstacles visualized in gray.

of up to 11.1s.

Example Fig. 6 shows an example environment and the solution obtained by our proposed framework. The high level policy is indicated in text, and the low level policy is visualized as a vector field for each obstacle. Only low level policies corresponding to obstacles that are chosen by the high level policy are visualized to preserve a clear view.

Comparison [4] We now weigh the benefits of our method to common heuristic approaches for object selection. Similar to [92, 93, 95] we implemented a constraint relaxed planner operating on $\mathscr{C}_r$. The planner performs an A* search over the discretized representation of $\mathscr{C}_r$ but considers collisions with movable obstacles as soft constraints rather than hard constraints. A heuristic cost penalty is applied for each initial intersection with a movable

[4]This set of experiments were performed on a different machine.

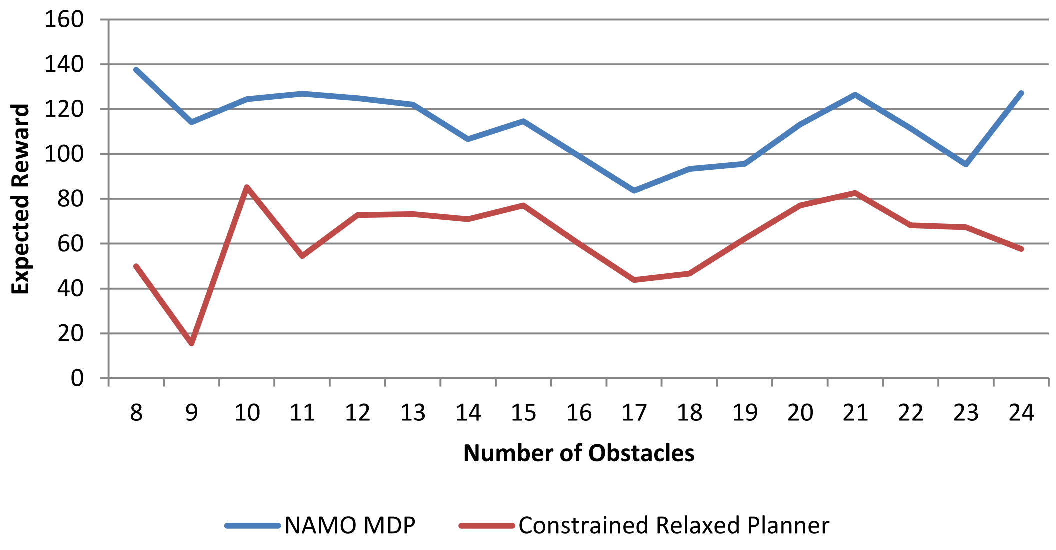


Figure 7: Average expected rewards for 300 runs.

obstacle. The A* search returns a motion path for the robot as well as a list of obstacles that have to be manipulated to clear the path. Assuming a deterministic displacement model for each action and object (we used the mode of Eq. 6), the planner then iterates through this list of obstacles and verifies that they can be manipulated to clear the path. If this fails for a specific obstacle, the planner marks the obstacle as a hard constraint and re-computes.

Analogous to previous NAMO planners, e.g. [92, 93, 95], this constraint relaxed planner does not consider the uncertainty the robot may have about the displacement model of a specific obstacle. The plans obtained by the constraint relaxed planner may consequently differ to the one obtained by our method. To evaluate this difference, we compared the expected rewards for the different plans. The expected reward for our method is given by the value of the NAMO-MDP state containing the initial robot configuration after convergence of value iteration. We computed the expected reward for the plan obtained by the constraint relaxed planner by iterating through the list of obstacles to manipulate in reverse order, determining the free-spaces the manipulation is effectively connecting and evaluating this action using a call to the MCTS planner. This equates to an interpretation of the solution obtained by the constraint relaxed planner as a partial policy in the NAMO MDP.

Consequently, the expected reward is again given by the value of the NAMO-MDP state containing the initial robot configuration. Fig. 7 summarizes the results obtained for 300 randomly generated environments (same generation method as described above). While in principle it is possible that the constraint relaxed planner chooses the same obstacles to manipulate as the proposed method, we see that on average our method substantially outperforms the constraint relaxed planner for all number of obstacles. The average expected reward over all 300 runs was 112.70 for the proposed method and 62.61 for the constraint relaxed planner. The average runtime was 43.41s for the proposed method and 10.62s for the constraint relaxed planner.

We observe that while the runtime of our method is higher, the expected reward is also substantially higher. This indicates that our method leads the robot to choose obstacles to manipulate that are more likely to create the desired free-space connectivity. The expected execution time, which typically dominates the computation times for both methods, as well as effort by the robot is consequently lower for the proposed method. In addition the constraint relaxed planning system cannot incorporate more information if they become available. The planner is deterministic in the objects it chooses for a given start and goal configuration, independent of the level of uncertainty the planner has about the objects. Giving the constraint relaxed planner more time or information will not change the selected objects.

The empirical results are consistent with the theoretical analysis of our algorithm, and support applicability of the approach to discretized environments. However, it may not always be possible to discretize a realistic robotic system without sacrificing resolution or runtime outside a reasonable scale. The following section will therefore present an alternative method for constructing and solving the NAMO MDP applicable to continuous environments.

3.4 Continuous Environments

Although the hierarchical MDP formulation in Section 3.3 is well-matched to the NAMO task, several of the techniques presented in Section 3.3.3 are not applicable for continuous state and action (control) spaces.

Most significantly, the subtasks can no longer be modeled as a MDP because their state and action spaces are no longer finite: on a constraint rigid body, every grasp point and force vector constitutes a unique action, and produces a real-valued displacement to arbitrary precision. As a consequence, a Q-function is no longer an appropriate policy representation, and seeded-wavefront can no longer be used to identify free-space regions. While discretization is possible in principle, our focus will be on methods that are compatible with the feedback control stacks and sampling-based planners used in practice on state-of-the-art robots.

The central challenge is to model uncertainty in object manipulation in a way that reflects the underlying dynamics of the robot-obstacle system. To do this we introduce object physical properties, such as mass and friction, to parameterize their rigid-body dynamics. These properties are not observable to the robot, but can be inferred from interactions in the environment. Inference of unobservable dynamic parameters is an important topic in its own right, but is beyond the scope of this work and we refer the interested reader to [84] for a detailed discussion about inference of dynamic parameters. However, the core observation is that from the robot's standpoint, these dynamics remain stochastic, arising from the interaction between a given controller and robot's *beliefs* about the physical properties of the target object.

In this section we therefore present a second method for constructing and solving the NAMO MDP building on a realistic manipulation stack. The goal is to achieve decision-theoretic planning in the NAMO task for *dynamically situated* robots. Here the term "dynamically situated" refers to both the existence of continuous state and control spaces, as well as the customary stack of closed-loop controllers and planners which have been developed to accommodate these situations. To accomplish this we represent transition uncertainty close to its source as object property uncertainty and perform Monte Carlo Simulation over the resulting object property distributions to obtain estimates for R_h and T_h.

After formally defining Monte Carlo simulation, we first describe how we can obtain the NAMO MDP abstraction in continuous environments before introducing the domain uncertainty representation in detail. Thereafter, we demonstrate and evaluate a concrete implementation.

3.4.1 Preliminaries

3.4.1.1 Monte Carlo Simulation

Monte Carlo simulation is a statistical tool relying on the law of large numbers to numerically solve problems that are difficult or impossible to solve analytically. To obtain an output measure for a function $f = H(\mathbf{X})$ with dependent parameters $\mathbf{X}$ governed by a probability distribution $P(X)$, Monte Carlo simulation procedes by repeatedly sampling a specific vector $X \sim P(X)$ and obtaining the output value $f = H(X)$. Statistical inference is then performed on the obtained output values [54]. This is similar to, though more general than Monte Carlo Tree Search, and can be used to approximate object manipulation models with arbitrary dynamics H and properties x.

3.4.2 Continuous NAMO MDP Construction

3.4.2.1 State and Action Space

The NAMO MDP relies on representing the continuous configuration space as an abstract MDP. Consequently, it is necessary to identify the independent free-space regions and their associated actions in the continuous robot configuration space. For specific environments specialized methods can be utilized to accomplish this. For example, in a 2D or 3D configuration space with polygonal objects a visibility graph planner [13, 70] could be utilized to deterministically determine the disconnected regions. Alternatively, for low dimensional spaces it may be possible to discretize the space and re-use the wavefront approach described in Section 3.3.2.1. However, as shown in [52], producing reasonable behavior on real robots would require far too many states. We therefore adopt a sampling approach that has been proven in the planning literature which involves building a *roadmap* over the state space. A roadmap is a graph representing the collision-free connectivity of the state space [52]. The main insight behind this approach is that the resulting roadmap will contain disconnected sub-graphs for precisely the free-space regions that we are attempting to identify.

3.4.2.2 PRM State Clustering

The probabilistic roadmap (PRM) [42] algorithm samples random configurations within the configuration space and connects nearby collision-free samples if there is a collision-free path between them. This algorithm can be viewed as an approximation of the wavefront method by considering only a set of randomly sampled states. Constructing a PRM in a disconnected configuration space will consequently return multiple disconnected sub-graphs. Each of these sub-graphs now encodes a separate free space region and together they yield the high level state set S_h.

Having determined S_h, the action set A_h needs to be determined. This is done by finding the obstacles that disconnect free-space regions. We accomplish this with a slight extension to the PRM construction phase: Instead of only considering random state-space samples during the construction of the PRM, we also use samples of valid grasping poses for objects. If, upon termination of the PRM construction phase, sampled grasping poses of the same obstacle belong to different sub-graphs, the obstacle is considered disconnecting the free-spaces and an according action is added to A_h.

3.4.2.3 Model Representation

To generalize NAMO to continuous domains we require a dynamics model that is both general-purpose, and scalable to the configuration spaces typically found in natural environments. The encoding of domain uncertainty using a displacement model for objects as in Section 3.3.2.2 is indeed general-purpose, but only applies to *open-loop* action primitives. To support closed-loop controllers and state-space planners as are typically used in robotics, we require a model of the form $f(X,U) \rightarrow Y$, where X and Y represent the pose and velocity of the target obstacle, and U the instantaneous manipulation control vector. Importantly, f must be compact, such that it can model the dynamics of an entire NAMO task with a number of parameters that can still reasonably be specified by a designer or learned from data.

3.4.2.4 Compact Stochastic Dynamics

Our approach is to represent manipulation uncertainty "close to its source" as uncertainty over the parameters of the underlying physical model of the robot-object system. We capture this uncertainty using distributions over the relevant physical quantities, such as masses and friction coefficients, and obtain transitions by sampling outcomes of a physical simulation defined over those random variables.

To motivate this approach, as opposed to classical regression-based alternatives, consider the problem of modeling a single table of unknown mass which may or may not have wheels. Assume the robot can apply controlled forces and torques through an effector equipped with a 3-axis force-torque sensor. We wish to model this table's 2D dynamics as a function of the applied forces:

$$f(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}, f_x, f_y, \tau) \rightarrow (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) \tag{15}$$

If the table is assumed to be a simple rigid-body, then Y is linear in X and U, and f can be compactly estimated by linear-regression over $|X| + |U| + 1 = 11$ parameters. However, due to the possibility of wheels, the table is potentially non-holonomic, implying that f is potentially non-linear. The two popular regression models which are capable of handling arbitrary non-linear f are Locally-Weighted Regression [4] and Gaussian Processes [79]. However, both of these methods are *non-parametric*, meaning that the number of required parameters (data points) grows with the complexity of f. Fully modeling a wheeled-table, for example, would require storing a set of $\{X, U, Y\}$ for all possible control vectors relative to the table's state – a potentially vast number of data points. Furthermore, these points must be obtained empirically or defined by considering the raw outcome space, which is significantly more cumbersome than specifying a small number of intrinsic physical quantities such as table mass and wheel orientation.

As a result, these non-parametric methods are widely applied to apprenticeship settings where sufficient data is available. However, they are not appropriate choices for NAMO tasks containing a large number of possible objects. This is particularly true because NAMO planning typically involves interacting with only a subset of the possible objects, using a subset of the possible behaviors, neither of which are known a priori.

3.4.2.5 Parameterizing NAMO Physical Dynamics

For the NAMO task we require a physics model which can capture the space of behaviors for large home and office objects, such as tables and chairs. We start by defining Ω as the minimal set of quantities governing object dynamics on a plane, and include mass and one or more anisotropic (wheel) constraints. Importantly, Ω should be viewed as a set of auxiliary variables for predicting transition dynamics, and not a state parameter itself.

To understand this as a stochastic transition model, consider the special case of zero-uncertainty where $P(\Omega) = \delta_\Omega$, with δ_Ω denoting the dirac delta function on a particular assignment to Ω. Here the model reduces to a deterministic environment governed by rigid body physics and the chosen controllers. Consequently, the transition model reduces to 0 or 1 and the reward function becomes a deterministic function of the physical work required by the robot to manipulate an object to create an opening. For the general case given an arbitrary distribution over Ω (e.g. multivariate-normal), we may generate samples of the transition dynamics in the NAMO MDP by sampling object parameters from $P(\Omega)$. Repeating this process and performing statistical inference over the obtained transition and reward models represents a stochastic simulation with outputs that reflect the expected interaction between the possible physical properties and the robot.

Our particular choice of parameters aims to satisfy the LP_1 class of NAMO problems [92], where regions are separated by individual, disconnected obstacles. We therefore require parameters to describe rigid-body dynamics for non-coupled objects, such as carts, chairs, tables with lockable wheels and casters. The central aspect governing the dynamics of this class of objects is contact with the ground. This includes isotropic friction forces, such as table feet, and more generally anisotropic friction forces to accommodate wheels. Other types of constraints, such as prismatic and revolute joints, serve to couple multiple bodies, and remain a challenge for future work.

In 2D, an anisotropic friction joint between the ground and a point on the target object can be represented five new parameters per joint: $\{x, y, \theta, \mu_x, \mu_y\}$, corresponding to 2D pose and orthogonal friction coefficients. For typical wheels, one of these friction coefficients is close to zero, and the other close to one. Furthermore, two joints are sufficient to describe any wheeled body in 2D, because any two joints at a given orientation can be expressed as a single joint along their common axis, and more than two joints at unique orientations serve to fully constrain the system (we do not consider what happens when constraints are violated). This means that the transition dynamics of arbitrary disconnected obstacles in 2D, including shopping carts, wheelchairs, and tables with lockable wheels, can be fully specified with $1 + 5 \times 2 = 11$ parameters. The first parameter is reserved for mass, which affects friction forces as well as the overall effort of manipulation.

Table 1: Distribution parameters for each physical parameter type

	Distribution
m	log Normal(μ, σ^2)
μ_x, μ_y	truncated Normal(μ, σ^2, a, b)
x, y	truncated Normal(μ, σ^2, a, b)
θ	von Mises(μ, κ)

Of these, mass m can take values in $\mathbb{R}^+$, friction coefficients μ_x, μ_y can take values in $[0, 1]$, position parameters x, y can take values within the bounds of the object $[a, b]$, and orientation can take values in $[-\pi, \pi]$. To represent the robot's beliefs over these parameters, we assign the distributions summarized in Table 1.

All of the distributions we used for object properties were in the Normal family. Our specific choice for the position, friction, and orientation parameters was necessary to constrain their support to legal values. For mass we required a distribution over $\mathbb{R}^+$, and chose a log Normal rather than a truncated Normal to reduce the number of parameters.

The full object property model is an assignment to these $2+2\times(4+4+2) = 22$ parameters for each object. Note that the solution approach merely requires a generative model for the relevant physical quantities. The distributions and or parametrizations are likely to be adapted for the system at hand. Object categories can be exploited to reduce redundancy and speed the of parameter inference. While we have not yet taken advantage of this property, it is an exciting topic for future work.

3.4.3 Solving the NAMO MDP

Using the object property model, it is possible to solve the NAMO MDP for dynamically situated robots. While the solution method for discretized environments in Section 3.3 obtained estimates for these quantities by introducing and sampling a low-level MDP, our solution method for continuous environments builds on Monte Carlo simulation to obtain estimates of T_h and R_h directly. The trade-off is that direct simulation discards all intermediate results, and maintains no policy information for visited states.

To see how estimates of T_h and R_h can be obtained from Monte Carlo simulation, let $T^a_{ss'}$ and R^a_s represent the specific instance of the transition and reward function for action a and states s and s'. To obtain value estimates, we perform the following evaluation k times[5]:

1. Sample world w with object parameters $\omega_i \sim P(\Omega_i)$ for all objects.
2. Call a manipulation planner on w trying to create an opening between the free-spaces represented by s and s' using the object represented by a and save the result.

$T^a_{ss'}$ is now set to be the ratio of manipulation plans that succeeded in creating an opening.

$$T^a_{ss'} = P(s' = target|s, a) = \frac{|succ|}{k} \tag{16}$$

R^a_s is set to:

$$R^a_s = \frac{1}{|succ|} \sum_{s \in succ} \alpha F(s) + \beta \tau(s) + \gamma T(s) \tag{17}$$

where *succ* denotes the set of successful manipulation plans, $F(\cdot), \tau(\cdot), T(\cdot)$ functions returning the force, torque and time required by a specific plan respectably. The constants α, β, γ represent weights. Given these estimates, M_h can be solved using standard value iteration. Algorithm 2 summaries the solution approach and the following section demonstrates a concrete example implementation.

3.4.4 Example Implementation

While the discussion so far has presented the general approach, we now present a specific example implementation. The main implementation challenge lies with the Monte Carlo simulation.

[5] k may be a fixed value or a function of the degree of uncertainty

Algorithm 2: Proposed framework for dynamically situated robots.

```
    Input: $W$: world, $k$: number samples for Monte Carlo simulation, $c_g$: goal
1   while robot not at $c_g$ do
        // determine MDP:
2       $MDP \leftarrow$ GET_STATES_AND_ACTIONS($W$); // perform value
            iteration:
3       while something changes do
4           for $s \in S$ do
5               if s contains goal then
6                   $V(s) \leftarrow$goal reward;
7                   continue;
8               $V(s) \leftarrow 0$;
9               for $a \in getActions(s)$ do
10                  $s' \leftarrow getTargetState(a)$;
11                  $trials \leftarrow []$;
12                  for $i \leftarrow 0$ to $k$ do
13                      $w \sim P(\Omega)$; $trials[i] \leftarrow$CONNECT_FS($s, s', a$);
14                  $succ \leftarrow \{$succesfull plans $\in trials\}$;
15                  $T(s,a,s') \leftarrow \frac{|succ|}{k}$;
16                  $R(s,a) \leftarrow \frac{1}{|succ|} \sum_{s \in succ} \alpha F(s) + \beta \tau(s) + \gamma T(s)$;
17                  $q_a \leftarrow R(s,a) + \gamma \sum_{s' \in ST(s,a,s') V(s')}$;
18                  if $q_a > V(s)$ then
19                      $V(s) \leftarrow q_a$;
20                      $\pi(s) \leftarrow a$;
        // perform actions:
21      $s_r \leftarrow getState(robot.config)$;
22      $robot.navigateTo(\pi(s_r).graspPos)$;
23      $robot.executeManip(\pi(s_r))$;
24      $updateBelief()$; // see [84] for details
```

To obtain estimates of $T^a_{ss'}$ and R^a_s, it has to be determined if, and at what cost, an opening can be created given a sample of $P(\Omega)$. To allow planning with arbitrary constraints on the obstacles and robot, we implemented a kinodynamic-RRT (RRT_{KD}) planner [53]. RRT_{KD} operates in the phase space of the robot – configuration and velocity components for each degree of freedom – and searches by sampling its control space. The RRT_{KD} search terminates if either a number of maximum nodes m have been reached or an opening has been created. Typically m should be small because the creation of an opening is usually a very local manipulation. Also, note that m reflects the computational resources spend on trying to determine the specific opening between the free-spaces represented by s and s' using the object associated with a. Similar to the dynamic horizon introduced for the discrete environment case in Section 3.3.3 it can be set dynamically to reflect the value of the target free-space. This allows us to take the value of creating such an opening into account and help to focus the computation resources to specific, important openings.

Recall that the NAMO task does not specify specific target locations for obstacles, but simply the creation of a free path. Consequently, the goal test function of the RRT_{KD} has to verify the existence of such an opening. Unfortunately, the verification of an opening is a non-trivial task in itself, and is therefore only performed every t node expansions in our implementation. We perform opening verification by checking the existence of a path between the free-spaces using a low-dimensional RRT (RRT_{FP}) [50] with a low maximum nodes threshold that only considers the robot's footprint. The start configuration for RRT_{FP} is given by the robot's configuration in the node in RRT_{KD} that triggered the opening verification while the goal configuration is set to be a random configuration within the goal free-space. If a path is found, an opening is reported. The astute reader will observe that openings might also be gleaned from the roadmap graphs used to determine the free-space regions. However, this is only possible if the graphs are maintained in synchrony with the manipulation planner, which we determined to be too expensive to use in practice.

Setting $T^a_{ss'}$ and R^a_s according to Eq. 16 and 17, the NAMO MDP can now be solved using value iteration. The following section presents an evaluation of this approach and example executions.

3.4.5 Evaluation

We provide a general complexity analysis of the proposed framework, show obtained run-times, and present example executions of our dynamic simulation. Recall that the difficulty of the NAMO domain is high-dimensionality caused by including obstacles in the state representation. We show that for the same conditions as in Section 3.3.4.1 (non-degenerate environments), we retain the linear dependence on the objects from the discrete case.

3.4.5.1 Theoretical Analysis

The proposed framework for continuous environments executes value iteration on the free-space MDP. Within each loop of value iteration we perform Monte Carlo simulation to evaluate reward and transition probabilities. The run-time of the Monte Carlo simulation depends on the number of iterations k and the chosen manipulation planner. In the proposed method, k is a constant independent of the sample space or distributions. While this does not suffice to provide general solution quality guarantees, it does decouple the complexity of the Monte Carlo simulation from the size of the state space. Therefore, as with MCTS, the manipulation planner provides constant-time solutions for the quantities required for value iteration in M_h.

As shown in Section 3.3.4.1, the complexity of value iteration is linear in the number of obstacles if the adjacency graph of the workspace is planar. The approach for the continuous environment therefore retains the same theoretical guarantees, including convergence, as the approach for discrete environments.

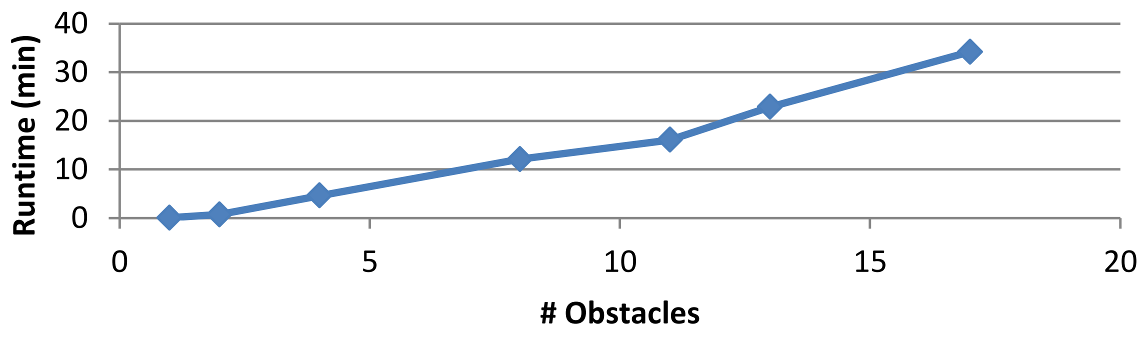


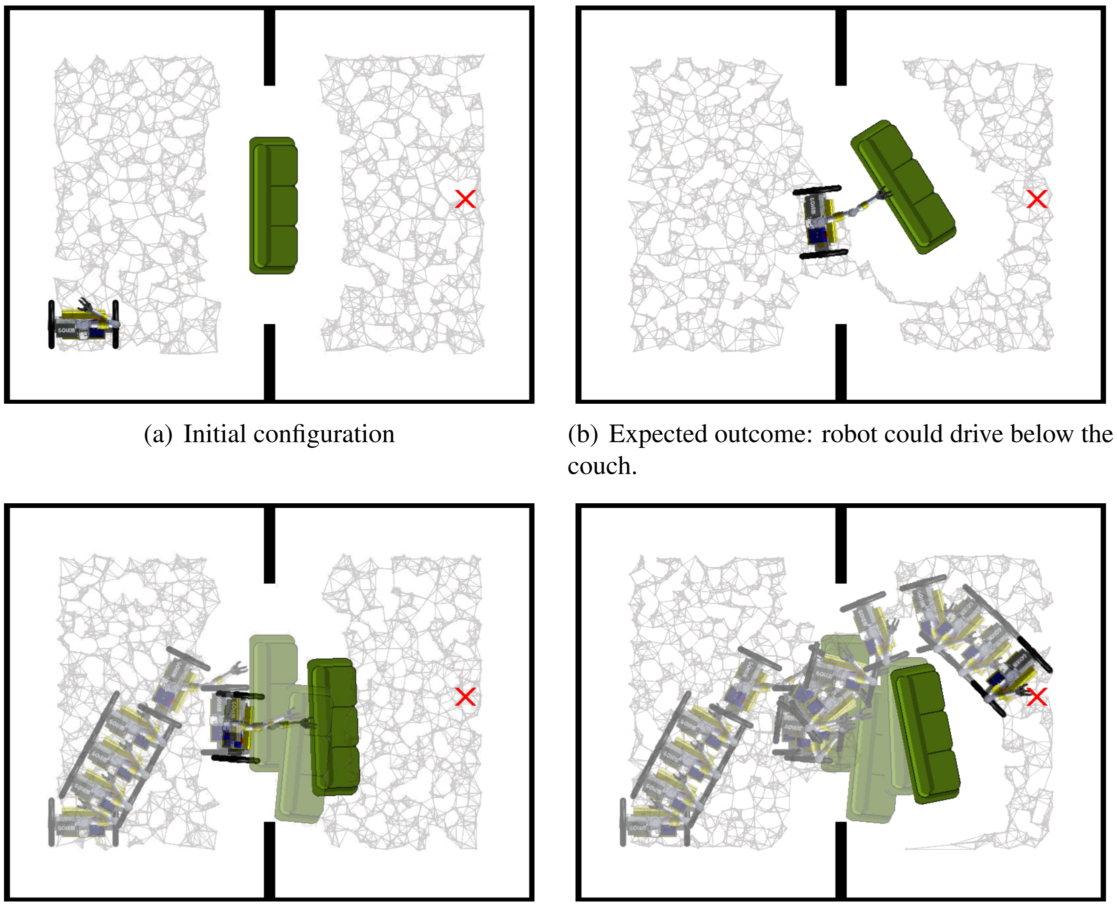
Figure 8: Average runtimes for 70 trials.

3.4.5.2 Empirical Analysis

We have implemented a prototype of the proposed framework in a simulation using Python and the open source 2D physics engine pyBox2D [2]. The robot is modeled as a nonholonomic 5 Degree of freedom (DOF) mobile manipulator with a 3 DOF base and a 2 DOF arm.

Runtime We performed 70 trials with varying environment sizes, objects, object placements, and object parameter distributions. For each trial, the distribution parameters of each object parameter were resampled from uniform hyper-priors. This was done to provide a method for automatically generating test cases across a range of valid belief states. Numerical examples for a specific case can be seen below.

The goal configuration was always in a different free-space region than the initial robot configuration. Fig. 8 summarizes the runtime as a function of the number of obstacles, again showing linear dependence. While the runtimes are in the range of minutes, orders of magnitudes of speedup could be achieved with more efficient programming languages and collision detection algorithms. Given the linear relationship, such constant factors present the bottleneck rather than the algorithm itself and are the subject of our future work. We now show some execution examples.


(a) Initial configuration

(b) Expected outcome: robot could drive below the couch.

(c) Actual behavior: robot recomputes based on new configuration.

(d) Moves couch further down and circumvents above.

Figure 9: Execution example of our dynamic simulation. The robot clears goal despite the couch behaving unexpectedly.

Example Executions Fig. 9(a) shows the initial configuration of a simple environment. The goal is defined with positive reward for reaching the free-space region on the right side of the map, but the couch blocks its path. In addition, there is uncertainty associated with the parameters of the couch. These parameters were defined in Table 2, and chosen to reflect uncertainty in the mass of the couch.

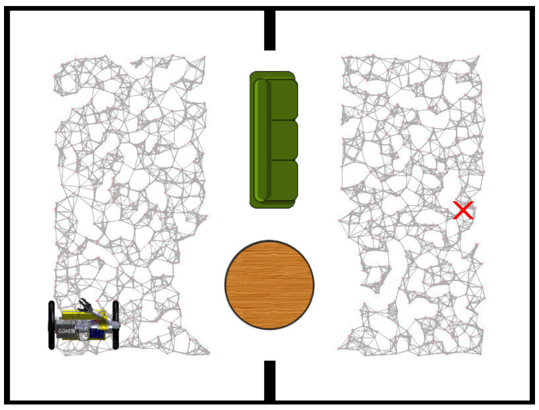
Table 2: Parameter distributions for the green couch.

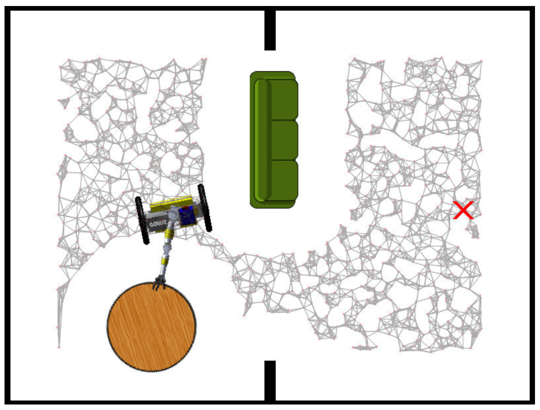
Parameter	Distribution
m (kg)	$\log \mathrm{Normal}(65, 15)$
μ_x	truncated $\mathrm{Normal}(0.05, 0.1, 0, 1)$
μ_y	truncated $\mathrm{Normal}(0.05, 0.1, 0, 1)$
x (m)	truncated $\mathrm{Normal}(0, 0.01, -5, 5)$
y (m)	truncated $\mathrm{Normal}(0, 0.01, -5, 5)$
θ (deg)	von $\mathrm{Mises}(0, 0.01)$

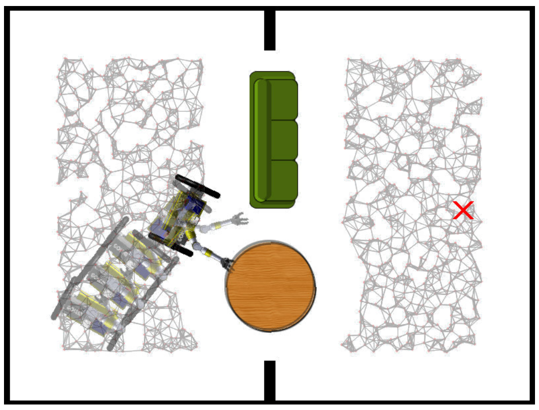
The robot executes our proposed framework, beginning by constructing a PRM over the space, shown in black. After determining the couch to be the only obstacle whose successful manipulation would clear the goal, the robot performs Monte Carlo simulation over manipulation plans for the couch. It then chooses a specific manipulation plan according to the likelihood of the object property sample that was used to construct the plan. The expected final couch configuration of the chosen manipulation plan is shown in Fig. 9(b). If the couch would indeed result in the expected configuration after the manipulation, the robot could circumvent the couch on the bottom and reach the goal.

However, because the exact object parameters are unknown to the robot, the execution of the plan instead results in the configuration shown in Fig. 9(c). Realizing that the required connectivity of the free-spaces has not been reached, the robot re-computes based on the new configuration. The robot decides instead to push the couch down a bit and move past it above. Fig. 9(d) shows that the robot now successfully clears the goal.

Fig. 10(a) demonstrates a similar setup but with multiple disconnecting obstacles. The robot again has uncertainty about the objects, but this time also begins with low probability that any of the objects are constrained. It decides to move the lighter table. The expected outcome of the chosen manipulation plan is shown in Fig. 10(b). However, as the table is in fact constrained to only rotate, the robot instead finds itself in the configuration visualized


(a) Initial configuration


(b) Expected outcome: table has low probability of being constrained.


(c) Actual behavior: table is constrained. Robot increases probability of table being rotationally constrained.

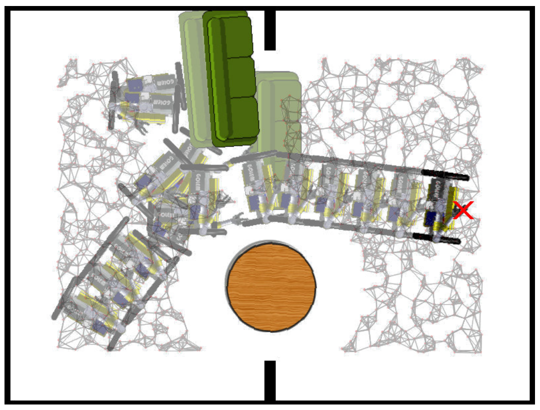

(d) Based on the new information the robot decides to move the couch.

Figure 10: Robot incorporates new information to enhance the plan.

in Fig. 10(c). The robot detects that, despite the applied force, the object has only rotated, not moved and updates its belief about the table being rotationally constrained. Given this new information, the subsequent plan involves moving the couch instead. This finally allows it to successfully clear the goal as visualized in Fig. 10(d).

While general inference techniques for updating the distributions given observed object behaviors is outside the scope of this work (see [84] for a suitable inference method), these examples show that our framework allows the robot to use updated belief distributions.

Fig. 11 shows an additional execution example with more than 30 obstacles.

These results demonstrate the usefulness of the proposed framework. The following

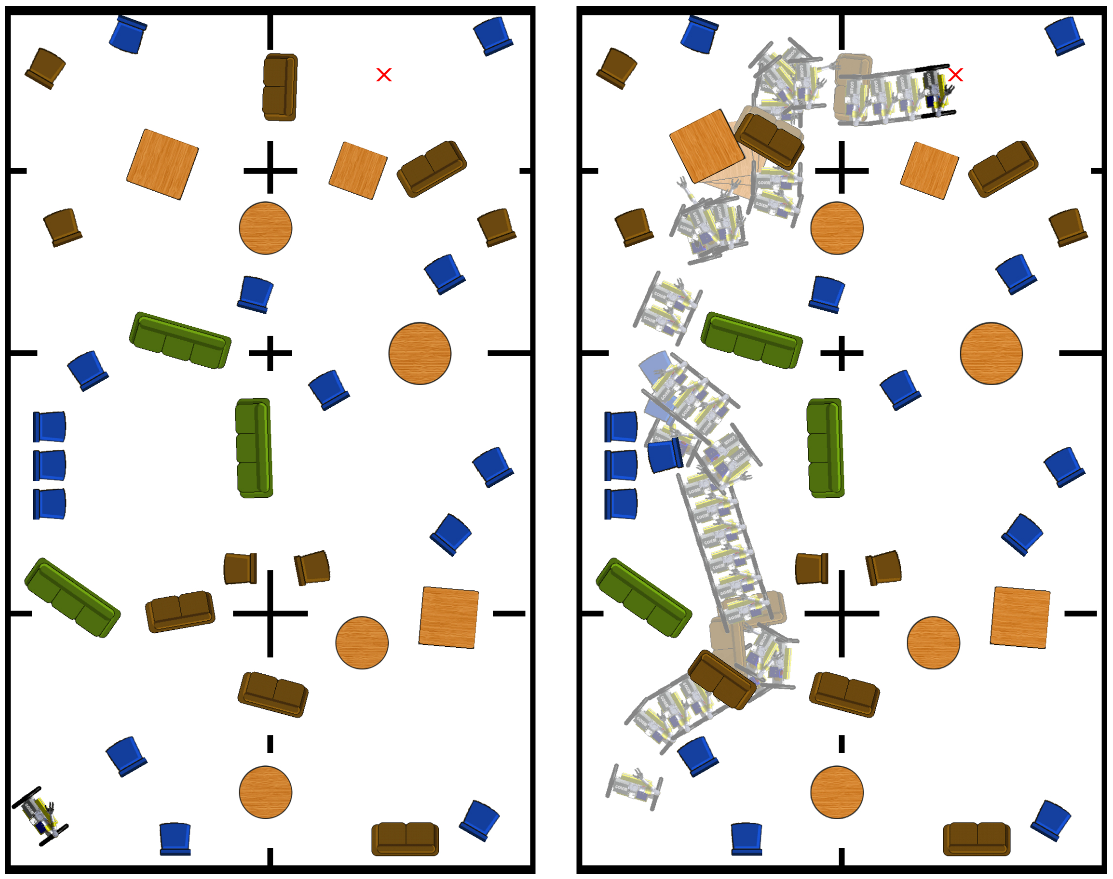

(a) Initial configuration. (b) Execution path of the robot.

Figure 11: Execution example of our dynamic simulation with more than 30 obstacles.

section provides online execution considerations for the NAMO MDP abstraction as well as general remarks about the framework.

3.5 Discussion

3.5.1 State-Space Change

If the robot successfully executes a low-level policy in its workspace, it, in general, has *merged* two free-spaces rather than transitioned between them. Further, the physical manipulation of environment objects with unknown dynamics could create new free-space regions by fragmenting the space. Consequently, a object manipulation could alter the state-space of M_h. A provably optimal planner should acknowledge this fact and directly reason about the resulting MDP changes. In other words, the planner should determine a "path of NAMO MDPs" that the robot transitions through on its way to the goal. While future work will investigate this concept as well as the applicability of policy iteration [81] to take advantage of the locality of the change, we address this challenge here by re-executing the proposed algorithm after each object manipulation. Given the linear runtime of the algorithm, this has not presented a significant disadvantage. However, as the algorithm does not consider all possible future free space configurations, the Q-values for a specific M_h do not represent the true long term expected reward of those regions, but rather an approximation based on the current world configuration. In general, this approximation is sound as long as actions are reversible. Future work will examine the expected loss and convergence properties under this assumption.

3.5.2 Summary

In this chapter, we have described the first decision theoretic formulation for the problem of Navigation Among Movable Obstacles. We introduced the NAMO MDP and provided techniques for constructing and solving the NAMO MDP in discrete and continuous environments. The NAMO MDP as presented here is applicable to domains where the robot needs to move a single object to connect two adjacent free-space regions.

Our abstraction and solution methods can be viewed from the top-down as a value-iteration scheme in the free-space representation of the NAMO problem. From this perspective, the primary difference with classical value iteration is that the Bellman updates issue calls to either a low-level MCTS planner for discretized environments or a Monte Carlo simulation for continuous environments in order to evaluate the action rewards, rather than querying an atomic reward function. We provided concrete example implementations for both cases. The general NAMO MDP framework, however, allows for other options than our specific choices. For example, the method presented in [55] could be used to evaluate the result of a manipulation action and potential existence of openings more efficiently. Exploration of alternative methods to the presented choices is of interest to us and will be a focus in future work.

In addition, while our framework handles multi-object interactions to some degree due to the online implementation of our algorithm, future work should investigate techniques to support more complex cases that require explicit multi-object coordination. We expect techniques similar to the regression planning presented in [95] to yield efficient methods for handling multiple bodies.

CHAPTER IV

NAMO WITH STATE AND ACTION UNCERTAINTY

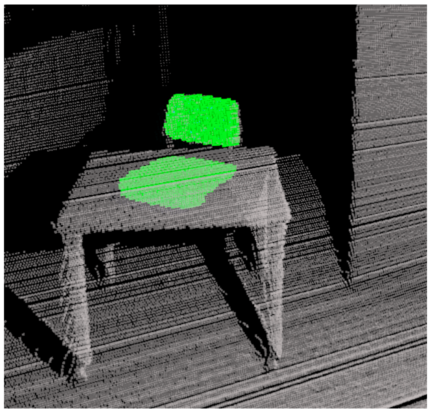
The previous chapter introduced the NAMO-MDP as a method for incorporating action uncertainty into the NAMO framework for scenarios in which the robot has access to positional information for all objects. However, in disaster areas or if the workspace is shared with other robots or humans, the robot might not have a-priori access to the location or even existence of movable objects within its path. Instead, the robot should autonomously detect environment objects such as furniture and reason online about which objects to move and where to. To bering robots closer to such capabilities, this chapter first introduces a vision algorithm for detecting the position of objects, even if they are partially occluded. We then use this capability and present a NAMO framework that can handle situations with substantial lack of initial state information as well as sensing and action uncertainty.

4.1 Detecting Partially Occluded Objects

To allow the robot to reason online about moving objects out of the way, it needs to be able to detect environment objects from limited sensory input, which is especially difficult if the objects are partially occluded. This section therefore presents a novel perception algorithm to obtain the position of objects even in the presence of occlusions[1].

In general, the ability to reliably detect and classify objects is crucial to the success of robots in realistic, human environments. Knowledge about the existence and position of objects within the robot's vicinity has many practical applications in robotics. For instance, semantic mapping, the creation of maps that include meaning, could be enhanced by detecting meaningful objects, such as furniture. A room with a table and many chairs is

[1]This work appears in [63].


(a) Example of a detected chair model, which is colored in green.

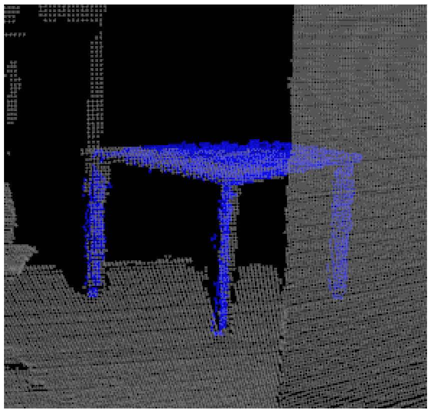

(b) Example of a detected table model, which is colored in blue.

Figure 12: Example of correctly classified furniture. The chair is correctly detected despite the fact that only the top part of the chair is actually visible in the scan. Similarly, the table is detected despite the fact that it is partially occluded.

likely to be the dinning room. In addition, knowledge about objects is crucial for robots to understand natural language commands such as ”put the mug on the table”.

Typical NAMO environments, as well as all of these examples, require the robot to perceive and reason about a human environment. Conditions in such environments are not ideal for a robot: objects are in different configurations at different times and are often partially occluded by other objects, walls or people. Any algorithm trying to reliably detect objects in human environments must therefore not just be able to handle arbitrary orientations of the objects but also occlusions.

Previous work has shown success in classifying mostly unoccluded objects in different orientations but does not perform well with the large occlusions typical for human environments. This section presents the Verfied Partial Object Detector (VPOD), an algorithm that is capable of detecting objects despite more than 50% occlusion. VPOD segments a point cloud of a scene into clusters by point distances and classifies each resulting cluster. This classification is based on a two step approach. The first step builds upon Viewpoint Feature Histograms (VFH) [82], a descriptor for 3D point cloud data that encodes geometry and viewpoint. The VFH of the query object is computed and compared to VFHs in a database composed of both complete object models and auto-generated partial object models. The auto-generated partial models allow for a classification of parts of an object, whose detection in turn yield a hypothesis of the existence of the complete objects in the point could. For example, detecting the back of a chair in the point cloud could indicate the existence of a chair with the sitting surface occluded by other objects.

However, the existence of partial objects in the database also increases the risk of false positives. For example, a simple piece of board could be matched to the back of a chair, leading to a hypothesis of a chair. Our algorithm therefore introduces a second step to verify each hypothesis. The verification step maps a complete model of the object associated with the candidate match into the point cloud of the scene. Points of the complete model that would be occluded by objects in the point cloud are then detected and eliminated from the model. The remaining points are checked for matching points in the point cloud. The proportion of matching, unoccluded points yields the final classification score. This steps ensures that matches are consistent with our expectation given the context of the world and the current viewpoint.

We now briefly summaries the core insights of VFHs and then discuss each of the algorithm steps in more detail.

4.1.1 Existing Method for Unobstructed Objects

Viewpoint Feature Histrogram are histograms describing the geometrical relationship between all points in the scan of an object. Rusu et al. [82] show that VFHs can discriminate according to the structure of the entire object, if the entire object is visible.

However the VFH is sensitive to partial occlusions. Consider the scan visualized in Fig. 12(a) in which only the top part of the chair is visible. As the top part of the chair is roughly just a flat surface, the points have an entirely different geometrical relationship to each other than all the points for an unoccluded chair. This results in different VFHs for the complete and partial chair, as shown in Fig. 13.

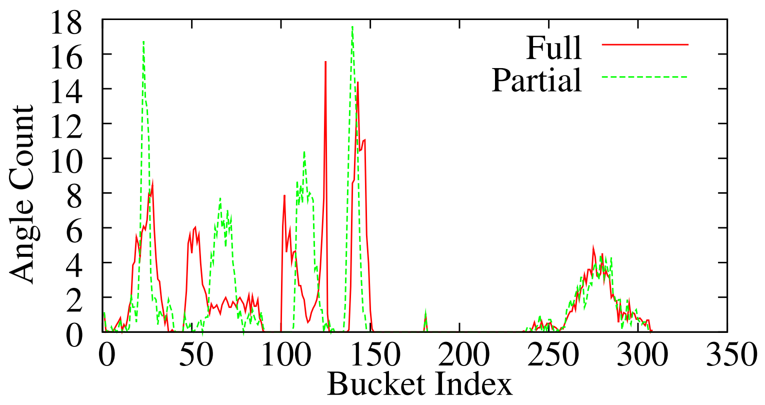


Figure 13: Example VFHs for a partial model and its associated complete model. The individual components of a VFH can be found in [82].

The consequence is that partially occluded objects are not reliably detectable with VFH or any other method that works with the properties of the complete object model alone.

4.1.2 Algorithm

We now outline our approach extending on the use of VFH.

4.1.3 Approach

The key insight is that fragments of a partially occluded object can be treated and classified as objects themselves. VPOD extends the VFH database to include partial models auto-generated by occluding portions of complete models. The partial model generation process is designed to generate typical occlusions occurring in the world. For human environments, we assume that the objects are usually occluded from one side (e.g partially behind a wall) or the bottom (e.g. a chair underneath a table). Our algorithm therefore generates partial models out of every complete model by successively removing points from each side of the object and from the bottom independently. This is done for a step size s and continued until a threshold t is reached for the remaining object size on the side currently affected by the removal. Fig. 14 visualizes an example of this process. Other types of common occlusions can easily be added by generating additional partial models. The resulting partial models are included in the VFH database *with* the complete models.

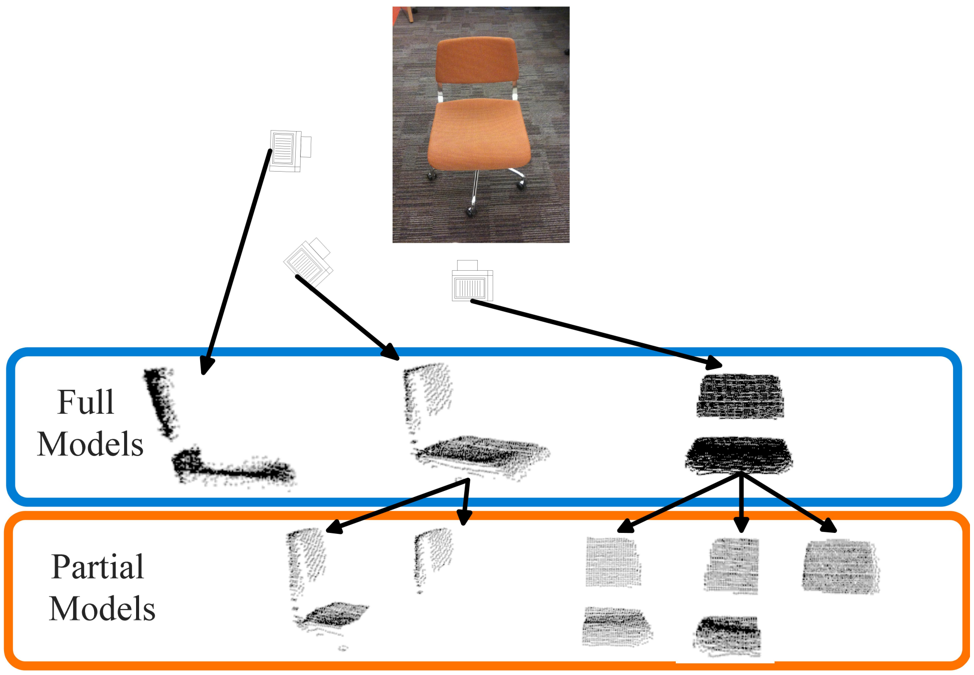


Figure 14: Hierarchy of complete models and auto generated partial models.

Including partial models in the VFH database increases the likelihood that a partially occluded object will be matched. For example, it is now possible to match the back of a chair against models in the database that represent just the back of a chair. But this also increases the likelihood of false positives, as the partial models are less distinctive (e.g. the back of a chair is mostly flat). Extending the database to artificially created objects, especially pieces of objects, allows for matches of arbitrary objects. To compensate for this, the algorithm includes a verification step that verifies the candidate classifications against the actual scene in which the point cloud was captured (called the "world" from here on).

The intuition behind this verification step is that if the detected part is indeed a proxy for the actual object in the scene, we can compute our expected observation for the complete object. This can be done by projecting the complete object model on the detected part and reasoning about expected occlusions, which can be determined by a simple line-of-sight test against the scan of the world. The obtained expected observation can then be matched with the actual observation. For example, if we matched the back of a chair against an object in the world, then the expected observation of the complete chair model projected into the scan has to be consistent with the world.

We have implemented this intuition through the following steps:

1. The complete model associated with the detected part is mapped into the world.
2. The portions of the complete model expected to be occluded based on information provided by the world are removed from the model.
3. The remaining model points are checked for matches in the world.
4. Based on this score, the classification is rejected or verified.

Fig. 15 summaries the workflow as discussed in detail in the following.

Clustering First, the point clouds collected by the robot are segmented into query clusters. We filter the point cloud to remove outlying points that are not near any surface or object. This type of noise is especially common on occlusion boundaries with laser range finders. The remaining points are then downsampled to reduce computation time. A grid size of 1cm was used for the downsampling in this work. Because the sensor's height relative to the ground is known, we can easily remove the ground plane by discarding any points below a given z-coordinate. With the ground plane removed, we can then cluster the remaining points. In our implementation, we require a minimum of 100 points per cluster, and allow 10cm distance between points within a cluster. Objects such as chairs, tables, desks, walls of the room and others are returned as clusters. As we focus on furniture as a use case, we filtered clusters out that cannot represent furniture or parts of furniture. Filtering was done based on bounding box size and position to discard objects that are much too large, much too small, or not near the ground.

Alignment Next, VPOD computes the transformation matrix T_{center} that transforms the centroid of each candidate partial model M_p to the centroid of the cluster C it was matched

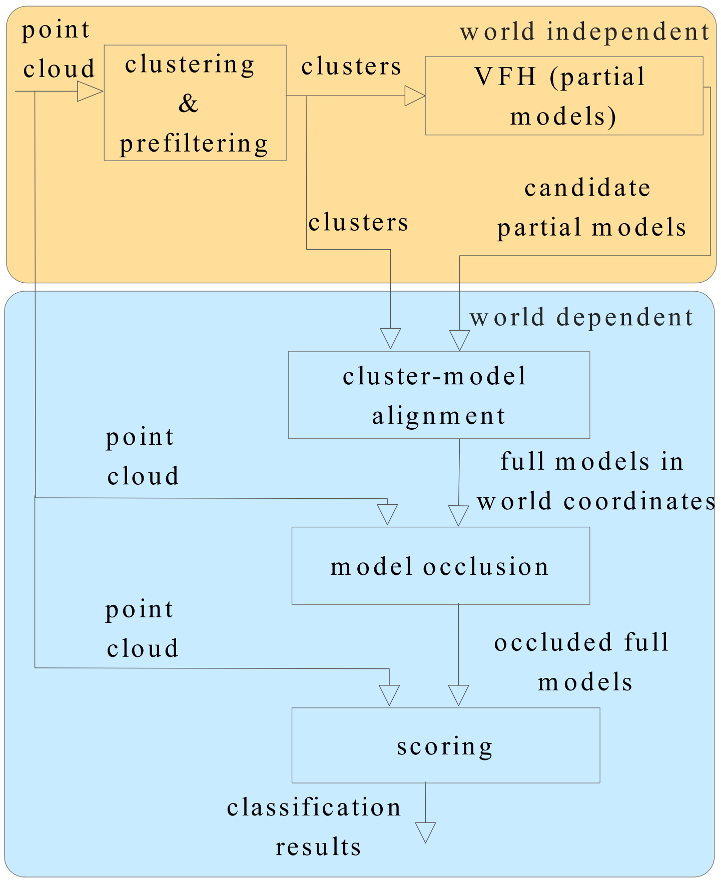


Figure 15: Workflow. Steps independent of the actual world are colored in orange while world dependent steps are colored in blue.

against. T_{center} is then apply to M_p. This yields the transformed partial model $M'_p = T_{center}M_p$. However, models in a database usually have limited angular resolution, *e.g.* we might have a model of a chair rotated at $10°$ and at $20°$ in the database and the transformation might consequently not align M_p and C optimally. In order to compensate for this, as well as centroid computation errors, Iterative Closet Point (ICP) [6] is performed on M'_p and C. The algorithm assumes that the objects are mostly upright and disqualifies candidate models if the rotation around the ground plane is exciting a threshold. The resulting transformation matrix T_{ICP} is saved. T_{center} and T_{ICP} are now both applied to the complete model M_c out of which M_p was generated yielding $M'_c = T_{ICP}T_{center}M_c$. Consequently, M'_c represents the complete model M_c mapped into world coordinates at the candidate location.

Occlusion Each point in the mapped complete model M'_c is now checked for occlusion. This is done through a technique based on ray casting. We adapted traditional ray casting to the property of a laser for which the lateral error typically increases with radial distance. As such, for each point p in M'_c, we check if any point in the full world scan W lies within a cone originating at the viewpoint origin and facing p. To check this easily, all the points in M'_c and the world W are transformed to radial coordinates around the viewpoint. The slope of the cone is tuned to match the known angular error of the laser. In order to compensate for noise in the radial distance, we require the occluding point to be a minimum distance in front of the occluded point. In addition, because a model should not occlude itself, we remove the currently considered cluster C from W prior to the occlusion test. Points that have been determined to be occluded in this way are omitted from M'_c. The resulting model $M^*_c = occlude(M'_c, W)$ is then scored.

Scoring The scoring of M^*_c is done by checking if each point p in M^*_c can be matched against a point in W. A point p in M^*_c is declared to have a match if W has a point within a small sphere around p. The final score is set to bo the ratio of matched points to total points in M^*_c. However, different scoring techniques are possible here. For example, an additional weighting factor determined by the number of points in M^*_c could be added to capture the lack of evidence that only a small number of points provides. Additionally the scoring could be done two-ways. This is, ensure that not just M^*_c coincides with W and C but that C also coincides with M^*_c. This would guarantee that most points in C have to actually be accounted for. Details are discussed below.

4.1.4 Experiments and Analysis

To verify our approach, we tested it on real scans obtained through a Hokuyo laser range finder on a Pan Tilt Unit.

We first created models of a chair and table for 7 distances and 16 rotations from scans with the laser range finder. The distances were chosen to yield a constant vertical viewpoint

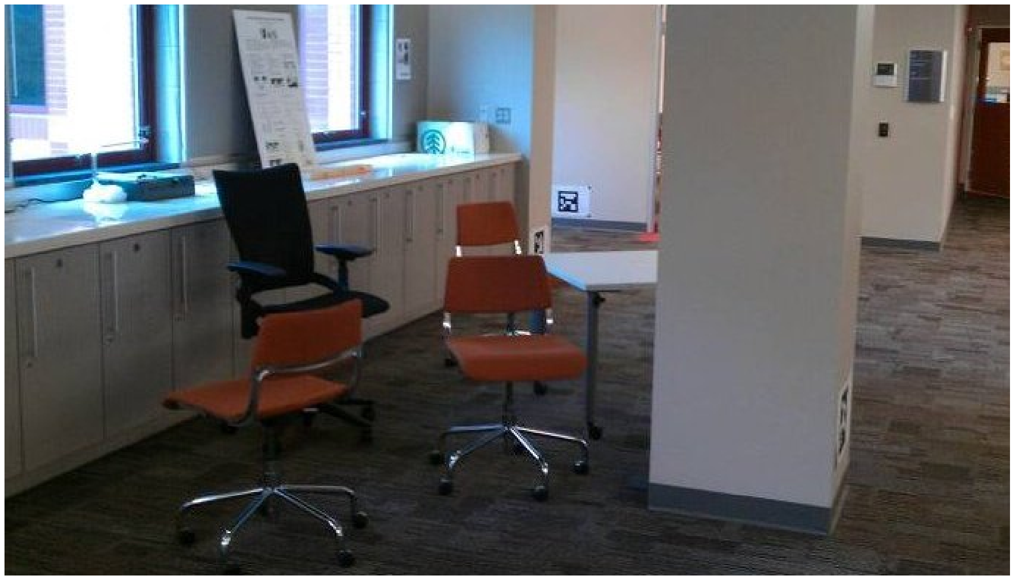
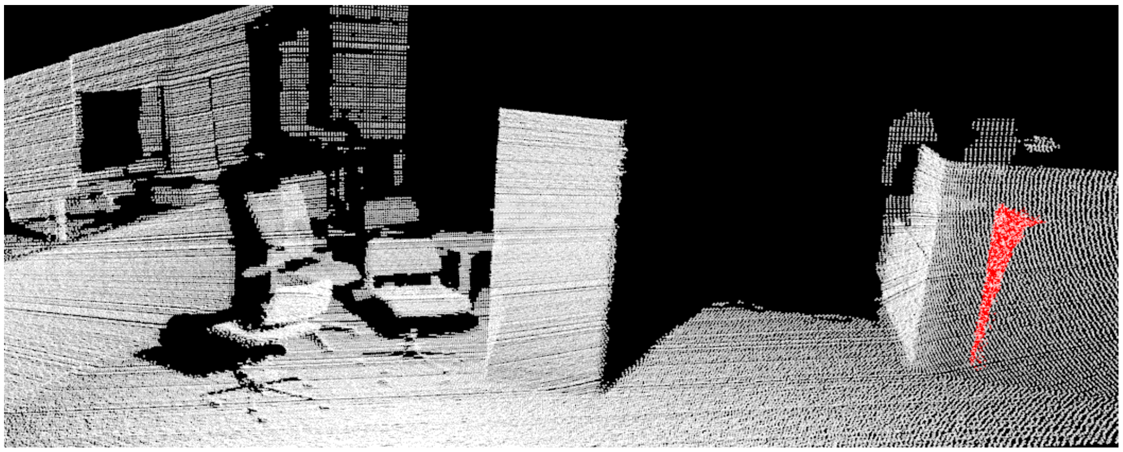
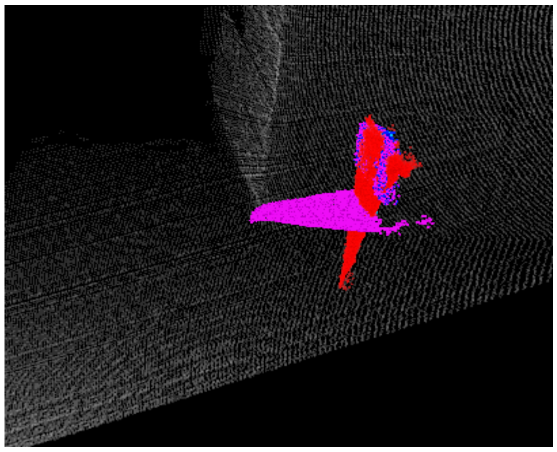


Figure 16: Experimental setup.

change of 9° on the obstacle. For our sensor height of approximately 1.5m, this resulted in distances of 0.86m, 1.09m, 1.35m, 1.66m, 2.06m, 2.59m and 3.37m. For each of the distances, scans of the object with a 22.5° rotational step size were obtained. This resulted in a total of 224 complete model scans. Out of those models, an additional 1068 partial models were auto-generated by consecutively removing points from bottom to top, left to right, and right to left from the model. We used a 10cm step size and proceeded until the remaining model had a size between 20-30cm on the axis currently affected by the point removal. Fig. 14 shows an example.

We took 30 test scans of scenes in an office environment with random configurations of up to four chairs of different types and two tables per scan. These scenes had non-furniture items, walls, unoccluded furniture, and partially occluded furniture. For example, some scenes had chairs behind and or pushed under a table. An example setup can be seen in Fig. 16 and the according point cloud is visualized in Fig. 17(a). As shown below, these are difficult cases for vanilla VFH which uses only complete models.


(a) Sample point cloud. Red: cluster classified by VFH as the top of a chair.


(b) False positive detection.

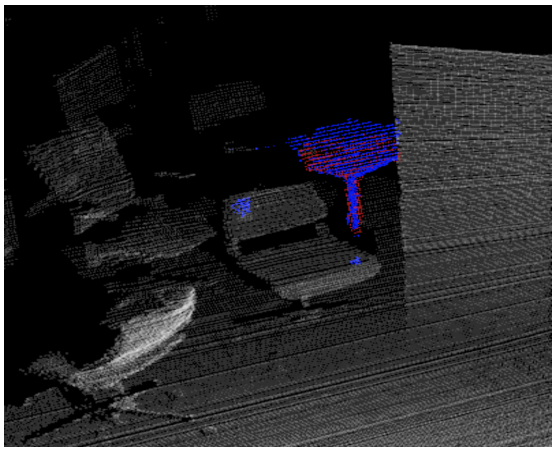
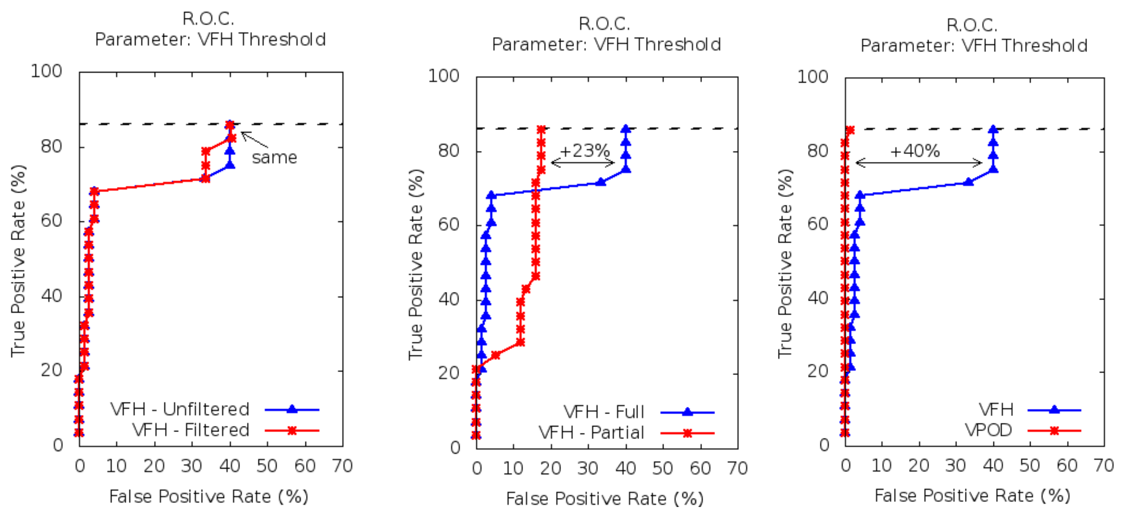

(c) True positive detection.

Figure 17: Example of false positive and true positive detection.

Results To obtain the following results we used the chi-squared distance metric for all comparisons between VFHs.

Fig. 18(a), Fig. 18(b), and Fig. 18(c) visualize the behavior of the classification error rates of different algorithms as the VFH threshold is relaxed. The True Positive Rate (TPR) denotes the ratio of actual furniture correctly identified; the False Positive Rate (FPR) the ratio of non-furniture incorrectly identified. Due to clustering errors explained below, not all objects in all test scenes could be classified by VFH or VPOD (or could be by any other algorithm that classifies based on clusters).

Fig. 18(a) shows the behavior of vanilla VFH using only complete models on the dataset



(a) Filtering improvement (b) Partial models improvement (c) VPOD improvement

Figure 18: Relative operating characteristic, showing improvement over VFH. Dashed line marks the best possible rate, due to clustering errors.

and the same algorithm with pre-filtering of clusters, as described in Section 4.1.4. Even if the VFH threshold is relaxed greatly, allowing *many* false positives, VFH is unable to correctly classify many clusters due to partial occlusions. Despite having more than 40% false positive rate, vanilla VFH was not able to achieve the maximal possible classification rate on our dataset. Even at a low threshold, vanilla VFH still produced a 5% false positive rate while only classifying 70% of the objects. Prefiltering provides only a modest reduction in the false positives produced by VFH. This sows that pre-filtering alone does not account for all of the benefits of VPOD.

Fig. 18(b) shows the behavior of VFH with and without partial models. At very low VFH thresholds, the partial models have no effect. As the threshold increases, the partial models are allowed to incorrectly match clusters, causing an increase in the false positive rate relative to complete models alone. But vanilla VFH using only complete models can only match the unoccluded clusters well and must allow many false positives in order to match the partially occluded clusters. However, VFH with partial models is able to classify all test clusters with a 23 point reduction in the FPR. This is because VFH with partial models can match partially occluded clusters using a tighter VFH threshold.

Fig. 18(c) shows the behavior of VPOD compared to VFH using complete models. Fig. 18(c) shows that VPOD, through the use of partial models, similar to Fig. 18(b), properly classified *all* test cases. Unlike VFH with partial models, VPOD classified all test cases with only a 1.3% false positive rate, due to the VPOD verification step. This allows VPOD to classify the partially occluded objects without introducing many false positives.

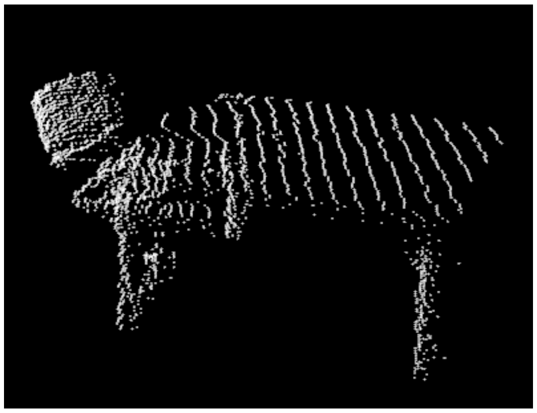
These results demonstrate the usefulness of VPOD.

Examples We disabled the pre-filtering and ICP restriction on y-axis rotation to obtain the following examples.

Fig. 17(b) demonstrates an example of the false positive detection. VFH classification with partial models classified the cluster marked red in Fig. 17(a) as being the top part of a chair. The algorithm then mapped the complete chair into world and scored it. Since the sitting surface is not occluded but also not visible in the scan this classification was rejected. In contrast, the table behind the chairs in Fig. 17(a) was matched by VFH with partial models and verified by the algorithm. Note that due to the occlusion, vanilla VFH using just complete models would not be able to classify the table.

Fig. 19(b) demonstrates a typical example where our algorithm fails if no pre-filtering or two-way matching is performed. If the cluster is unreasonably large, almost any furniture piece can be fitted in and have its points being accounted for. In Fig. 19(b) a chair (green) is being fitted into a big wall (fitted chair shown in blue). The algorithm is effectively saying that the chair is lodged in the wall. This demonstrates the necessity of pre-filtering or two-way matching in combination with our algorithm.

Clustering In our implementation, the clustering distance is set to a fixed value. This can yield results similar to Fig. 19(a) where multiple furniture pieces have been clustered together, which hinders classification. In future work we plan to have the clustering distance be data driven. The basic assumption is that points on the same object have a smaller relative distance to each other than points between objects. We therefore plan to evaluate


(a) Example of a chair and table being clustered together.

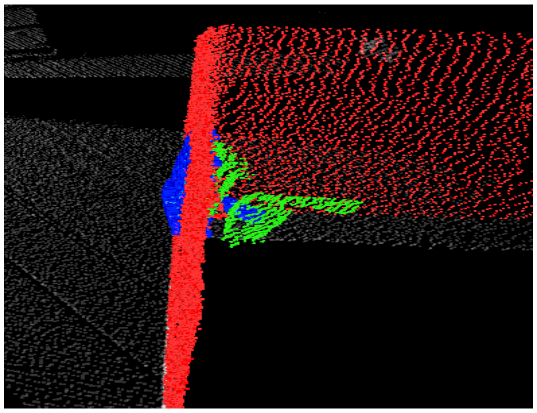

(b) Example of a small model fitted into a big cluster.

Figure 19: Examples of where our algorithm fails.

the distances of points within a cluster and re-cluster a cluster based on a new, smaller distance. Clusters that are not classified can be progressively partitioned and reclassified until either a match is found or the cluster size is unreasonably small.

Further, many objects such as chairs and tables can appear as multiple distinct parts due to self-occlusions. For example, when observing most standard office chairs, we typically observe the chair's seat, back, and wheeled base, but not the central supporting column due to the downward viewing angle. Similar results can also occur with other types of furniture. To address this problem, we anticipate that future iterations on our algorithm will project the clusters down to the ground plane, find the convex hull for each, and merge clusters that have overlap. This allows to consider such objects as one unit, despite being separated by a significant vertical distance. Again, this step can be verified by checking if the classification results have improved in comparison to the single clusters.

Runtime We are performing the occlusion and matching simultaneously in VPOD, which takes an average of 3.3 seconds for a world scan with about 32,000 points and an average cluster size of 19,000 points. If runtime is a concern the occlusion detection and matching function do not need to be done against the full point cloud. Rather, one should determine the relevant parts of the full point cloud affecting the current cluster and limit occlusion detection and matching to this subset.

4.1.5 Discussion

In this section we presented the VPOD algorithm for detecting partially occluded objects by matching clusters against segments of models and verifying our expectations against the world. VPOD extends the scope of vanilla VFH classification by using partial models. To reject false positives, VPOD verifies expectations about the predicted classification with the world. We verified the effectiveness of our approach on real data, showing improvement on cases not handled by VFH with complete models alone.

Given VPOD or similar algorithms, the robot can detect objects in its environment, even if they are partially occluded. The following section presents an algorithm that utilizes this capability to perform NAMO even without any initial knowledge of object locations.

4.2 State and Action Uncertainty in NAMO

We now build upon insights obtained by Chapter 3 and Section 4.1 and present a complete framework that allows a real robot to execute NAMO despite substantial lack of information about the initial state as well as action and sensing uncertainty[2].

A theoretically optimal strategy for such problems is to compute, offline, a policy that maps histories of observations into actions. This is computationally entirely intractable in large domains. A more computationally feasible strategy is interleaved online planning and execution, in which a partial plan is constructed online, using an approximate model for efficiency, the first action is executed, the belief state is updated, and a new plan is constructed. This has been demonstrated to be an effective approach in medium-sized discrete and continuous domains [76]. However, in very long-horizon problems, even finding an approximate plan in terms of primitive actions becomes computationally intractable.

[2]This work appears in [57].

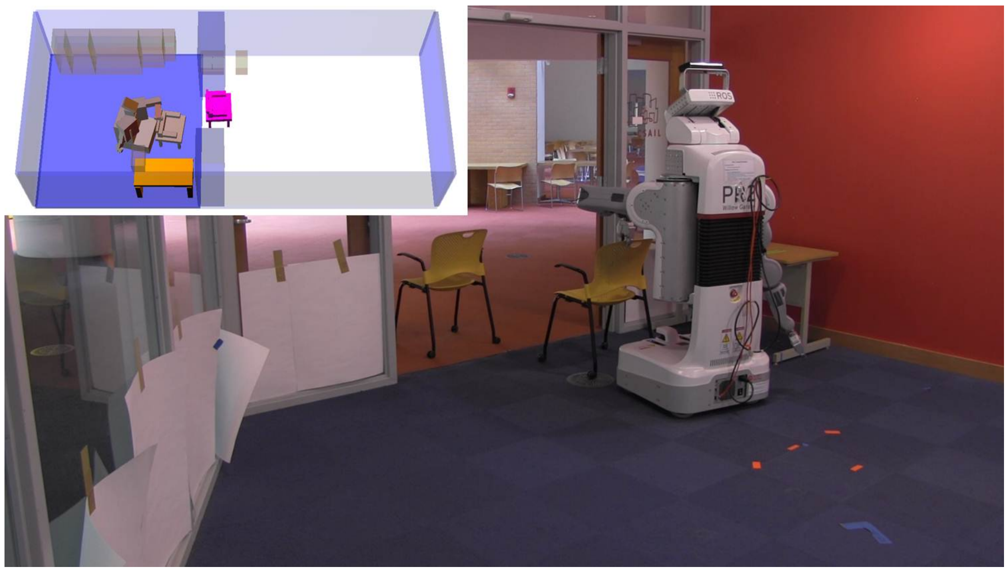


Figure 20: Solving a NAMO problem with a PR2. The robot discovers two unknown obstacles (chairs) and moves them out of the doorway in order to get through.

Previous work [39, 40] introduced a hierarchical strategy for interleaved online planning and execution, called *HPN* (Hierarchical Planning in the Now), that improved computational efficiency by using a temporal hierarchical decomposition of the planning problem into many small planning problems. In this section we demonstrate how HPN can be extended to yield a framework for NAMO domains with substantial lack of initial state information as well as sensing and action uncertainty.

In order to make *HPN* applicable to the NAMO domain, we introduce two general concepts and mechanisms that can be used to improve both efficiency and optimality for integrated task and motion planning and execution in large domains: **Reconsideration** and **Foresight**. We show that the hierarchical decomposition introduced in *HPN* to improve efficiency provides an opportunity to apply these broader concepts to improve optimality as well, measured in the cost of actions taken by the robot. Furthermore, our new techniques allow us to demonstrate and empirically evaluate the trade-offs between efficiency and optimality in simulation and on a real robot system.

Consider the following two concepts in the context of both efficiency (total computational time for all planning subproblems) and optimality (total execution cost of actions on the robot):

- **Reconsideration**: Deciding when to replan based on opportunity for plan improvement and computational cost of replanning.
- **Foresight**: Leveraging knowledge of future subgoals and beliefs about future observations to maintain correctness and improve optimality.

These concepts provide a general framework for understanding a wide variety of algorithmic strategies for improving efficiency, including the methods from the original *HPN* as well as a set of new methods for improving optimality that are introduced in this chapter.

Reconsideration in the original *HPN* was used to trigger re-planning and restrict it to a subtree of the hierarchical plan; it *only* triggered when the execution system was not able to achieve the preconditions of an action. This strategy led to computational efficiency but also to considerably suboptimality due to extraneous actions. We now observe that reconsideration can also be triggered to re-plan when *new opportunities* present themselves that are likely to improve the efficiency of the overall system.

Foresight in the original *HPN* applied only locally: within the planning process for a particular subgoal at a particular level of abstraction, the backward chaining algorithm computed conditions characterizing correctness requirements for the later part of the plan, which were used to constrain choices made during earlier parts of a plan. This degree of foresight was critical to guarantee plan correctness and improved computational efficiency by considerably decreasing backtracking.

In this work, we show that foresight can also be applied more globally, by taking into account the robot's intentions for future action, which are encoded in the hierarchical structure of subgoals created by the *HPN* process. Our new system not only ensures that early action choices do not prevent future success, but also tries to minimize the interference between separate subtasks of a plan.

This section describes our new techniques based on the concepts of reconsideration and foresight and identifies tradeoffs between efficiency and optimality in the context of these categories of reasoning. We first discuss these concepts generally before demonstrating how these contributions allow a real to robotic system to execute NAMO despite substantial lack of initial state information as well sensing and action uncertainty.

4.2.1 Existing Planning Framework

We now summarize an existing hierarchical planning and execution architecture, which will provide the basis for new foresight and reconsideration methods discussed below.

4.2.1.1 Hierarchical planning and execution architecture

The *BHPN* (Belief HPN) architecture [40] is a hierarchical planning and execution architecture designed to work in uncertain domains. It assumes the existence of:

- A *state-estimation* module which maintains a *belief state*—a probability distribution over the states of the world—reflecting the history of observations and actions made by the robot.

- A set of *operator descriptions* characterizing the operations that the robot can do in the world; each operator description has a set of preconditions and a set of effects, described in terms of logical predicates that characterize some aspect of the robot's belief about the state of the world. Abstracted versions of the operators, obtained by postponing consideration of some of the preconditions, are used to construct *hierarchical* plans.

- A *planner* that uses the process of *pre-image backchaining*. It starts from the goal, which is a description of a set of desirable states of the world, and works backward. On each step, it computes the *pre-image* of the goal under the selected action; the pre-image is the set of states such that, were the action to be taken, then the resulting state would be in the goal set. The pre-image is represented compactly as a logical formula. A call to the planner, starting in belief state b with goal set γ results in a list $((-,g_0),(\omega_1,g_1),...,(\omega_n,g_n))$ where the ω_i are (possibly abstract) operator instances, $g_n = \gamma$, g_i is the pre-image of g_{i+1} under ω_i, and $b \in g_0$. The conditions g_i play a critical role in the execution architecture: g_i is the set of states in which the plan $\omega_{i+1},\dots,\omega_n$ can be expected to achieve the goal.

Algorithm 3: Pseudo-code for the *BHPN* algorithm with reconsideration. In the default *applicable* and *nextStep* methods in Procedure 4 and Procedure 5 we assume that the plan p has an instance variable i that is initialized to 0 and used to keep track of the current step for execution of the plan.

```
Input: $b, \gamma, world$
$ps = $ STACK()
$ps.push($PLAN$PLAN([(None, True),(None,\gamma)]))$
while not $ps.empty()$ do
    $p = ps.top()$
    if not $applicable(p,b)$ then
        $ps.pop()$
    else
        $(\omega,\gamma') = p.nextStep(b)$
        if $\omega.abstract()$ then
            $ps.push($PLAN$(\gamma', b, p, \alpha.incr(\omega)))$
        else
            $obs = world.execute(\omega)$
            $b.update(\omega, obs)$
            RECONSIDER$(ps, b)$
```

Procedure 4: applicable

```
Input: $p, b$
return $p.g_{p.i} = \gamma$ or $(p.i < p.n$ and $b \in p.g_{p.i})$
```

Algorithm 3 shows the *BHPN* algorithm. It takes as inputs a current belief state b, logical goal description γ, and a world (real robot or simulation) in which actions can be executed. It manages a stack of plans, which are stored in variable *ps*; it is initialized to have a dummy plan, consisting of $g_0 = True$ and $g_1 = \gamma$, indicating that it can always be

Procedure 5: nextStep

```
Input: $p, b$
if not $p.g_{p.i} = \gamma$ then
    $p.i$ += 1
end
return $(p.\omega_{p.i}, p.g_{p.i})$
```

applied and that its overall goal is γ. The procedure loops until the plan stack is empty, which can happen only when γ has been achieved.

On each iteration of the algorithm, we examine the plan p that is on top of the plan stack to see if it is applicable. In the following sections we will consider a different definition of the *applicable* method, but we will begin with a very simple definition of it and the related method *nextStep*, defined in Procedure 4 and Procedure 5 respectively. In this case, p is applicable in belief state b if the plan has not reached the end ($p.i < p.n$) and b is in the preimage $g_{p.i}$ of the next step that will be executed, which is step $i+1$. If p is not applicable, then it is popped off the stack. Note that the initial plan is always applicable.

If p is applicable, then we determine the next operator ω and subgoal γ' by calling the *nextStep* method of p. The simplest version of *nextStep* ignores the current belief state b and simply returns the sequentially next step in the plan. If the next operator is abstract (*None* is always abstract), it means that its preconditions may not yet have been met in full detail, and so we construct a plan to achieve the subgoal γ' at a more detailed level of abstraction ($p.\alpha.incr(\omega)$) and push it on the plan stack. Otherwise, ω is a primitive, which is executed in the world, yielding an observation *obs*, and the belief state is updated based on that observation. After every action, we can (in principle) reconsider the plan; we discuss this in the next section.

It is important to see that a plan p may be popped off the stack for two very different reasons: (1) All plan steps have been executed with their intended results; or (2) Some plan step, when executed, did not have its intended result. In either case, it is appropriate to return control to the higher level (the plan deeper in the stack). If the plan executed successfully, then control will resume at the next step of the plan at the higher level of abstraction; if it failed to have the desired effect, then the parent operation will also fail, and so on all the way to the bottom of the stack, where the initial plan is expanded again. Later, we consider several extensions to make this control structure more robust and flexible.

Fig. 21 illustrates the nominal behavior of the BHPN algorithm, as a planning and execution tree. The blue nodes represent goals or subgoals; in this case, the overall goal, at the root node, is to have two objects in the cupboard. Goals and subgoals are characterized by conditions on belief states. The actual planning and execution tree is considerably more complicated: this example is abstracted to make it easier to explain its important aspects.

Each outlined box contains a plan. Within a single box, there are blue and pink nodes. Pink nodes represent operators and blue nodes subgoals. So, in Plan 1, using an abstract version of the *Place* operation, it is determined that, as long as both objects A and B are movable, a plan consisting of placing A, followed by placing B will achieve the goal. Note the intermediate subgoal: $\exists x.BIn(x, Cupboard) \wedge BMovable(B)$. That is the pre-image of the goal under the operation of placing B in the cupboard: as long as some other object is in there, and we additionally place B, then the goal will be achieved.

Plan 2 is a plan for achieving the first subgoal in Plan 1, which consists of picking up object A with grasp G and placing it in the cupboard. Plan 3 is a plan for coming to hold A in grasp G: it consists of a primitive *Look* action (shown in green), which is intended to achieve the condition $BVRelPose(A, Eps1)$, which says that we know the relative pose of the robot and object A to within some tolerance *Eps1*. This step ensures that the robot has a moderately good estimate of the pose of object A before trying to plan in detail how to pick it up. Plan 4 provides yet more detail, ensuring that the volumes of space (called "swept volumes") that the robot has to move through are clear, and that, before actually picking up the object, its pose id known to a tighter tolerance, *Eps2*.

The tree, as shown, is a snapshot of the *BHPN* process. Using the simplest *nextStep* definition, each plan would be executed left to right, completely planning for and executing

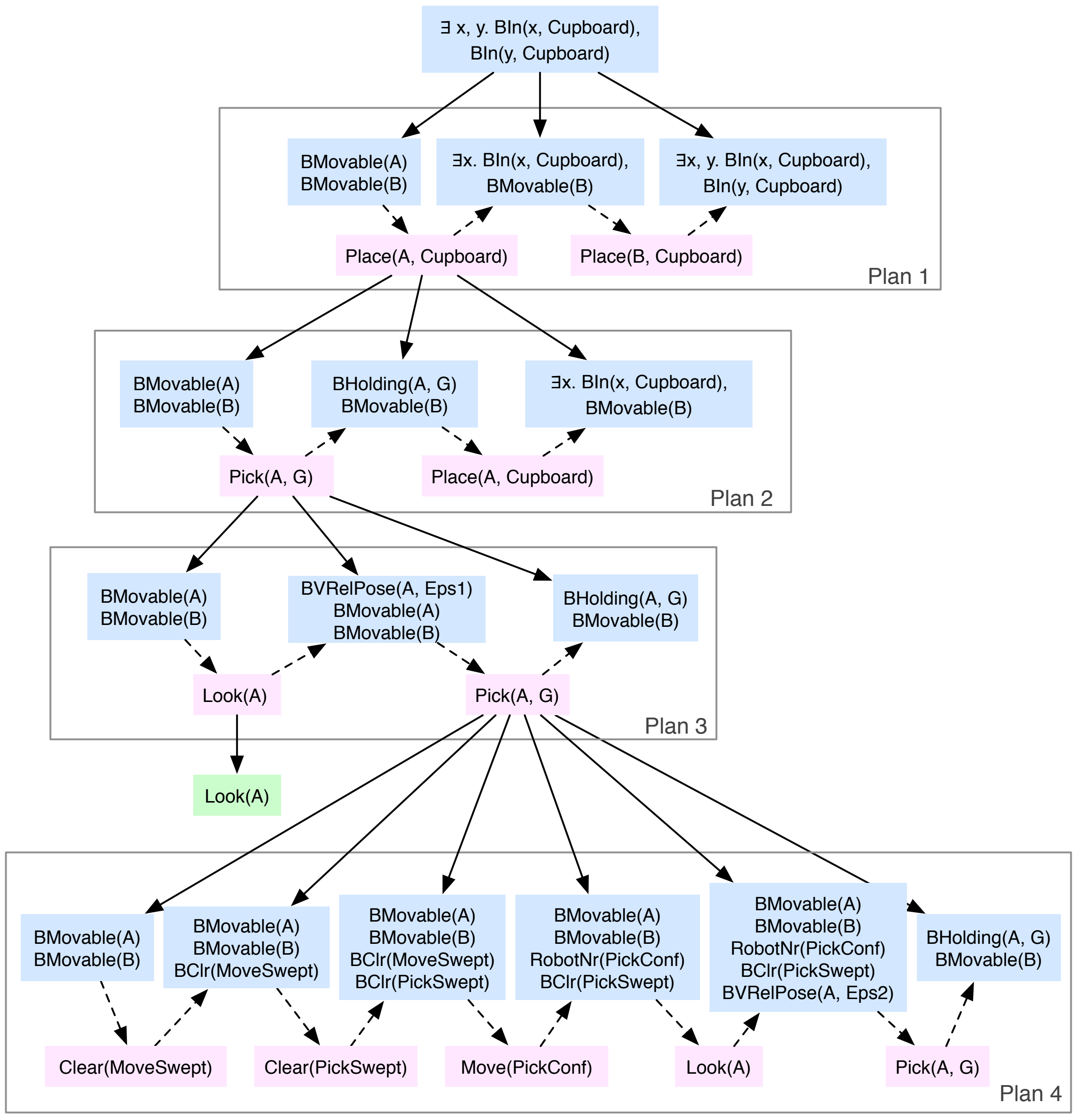

∃ x, y. BIn(x, Cupboard), BIn(y, Cupboard)
BMovable(A) BMovable(B)
∃x. BIn(x, Cupboard), BMovable(B)
∃x, y. BIn(x, Cupboard), BIn(y, Cupboard)
Place(A, Cupboard)
Place(B, Cupboard)
Plan 1
BMovable(A) BMovable(B)
BHolding(A, G) BMovable(B)
∃x. BIn(x, Cupboard), BMovable(B)
Pick(A, G)
Place(A, Cupboard)
Plan 2
BMovable(A) BMovable(B)
BVRelPose(A, Eps1) BMovable(A) BMovable(B)
BHolding(A, G) BMovable(B)
Look(A)
Pick(A, G)
Plan 3
Look(A)
BMovable(A) BMovable(B)
BMovable(A) BMovable(B) BClr(MoveSwept)
BMovable(A) BMovable(B) BClr(MoveSwept) BClr(PickSwept)
BMovable(A) BMovable(B) RobotNr(PickConf) BClr(PickSwept)
BMovable(A) BMovable(B) RobotNr(PickConf) BClr(PickSwept) BVRelPose(A, Eps2)
BHolding(A, G) BMovable(B)
Clear(MoveSwept)
Clear(PickSwept)
Move(PickConf)
Look(A)
Pick(A, G)
Plan 4

Figure 21: Illustration of *BHPN* approach

actions to achieve one subgoal before going on to the next. This results in a left-to-right depth-first traversal of the tree. If any operation, primitive or abstract, has a result that is not the expected one, then it will fail and control will return to the plan above it.

4.2.1.2 Flexible execution of existing plans

```
Procedure 6: applicable
   Input: $p, b$
1  return $b \in \bigcup_{i=0}^{n-1} p.g_i$ and $b \notin p.g_n$
```

```
Procedure 7: nextStep
   Input: $p, b$
1  $i^* = 1 + \arg\max_{i=0}^{n-1} b \in p.g_i$
2  return $(p.\omega_{i^*}, p.g_{i^*})$
```

The execution architecture described so far is very rigid: if a plan cannot be executed from left to right, then it is considered to be a failure. But, in many cases, the plan can be salvaged, and replanning limited.

Consider Plan 4. If the *move* action doesn't have the expected result because the robot undershoots its target, for example, then a reasonable course of action would be to re-execute the move operation. The same would be true if, upon executing the *look* action, it found that the variance on its pose estimate for A was not sufficiently reduced to satisfy the preconditions for the *pick* action.

We can generalize this idea further, to handle both *disappointments*, when an action does not have its desired effect and *surprises*, when an action actually results in more useful results than expected. Procedure 6 and Procedure 7 shows revised versions of *applicable* and *nextStep* that re-use a plan as much as possible. We say that a plan is applicable if any step of the plan is applicable. In other words, the current belief state b is in the pre-image of one of the operations, and the final goal has not yet been achieved. Thus, if an action results in a belief for which there is *some* action in the plan that is an appropriate next step, then the plan remains applicable and is not popped off the plan stack. In the *nextStep* method, we select the operation ω_i for execution where g_{i-1} is the “rightmost” pre-image that contains the current belief state.

In Plan 4, if the robot looks at A and finds that there is actually a previously unknown object intruding into the swept volume for the arm to pick the object, it could go back to the plan step that achieves $Clear(PickSwept)$ and continue execution from there. If it were to find, though, upon attempting to pick up A that it was, in fact, bolted to the table, the $BMovable(A)$ condition would become false. At that point, Plan 4 would be popped off the plan stack because none of the pre-images contain the current belief state. The same would be true for plans 3, 2, and 1 in succession. Replanning would be initiated for the top-level goal, hopefully finding a different object to place into the cupboard instead of A.

Another way to be flexible about the use of an existing plan is to re-order some of the steps, based on information available in the current belief state. We adopt the peephole optimization method of [29], which considers all of the plan steps with preconditions that are currently satisfied and uses a heuristic to decide which order to execute them in, or whether they are likely to interact very closely, in which case it will consider them jointly. In Plan 1, the peephole optimization may decide that it is preferable to put the larger object into the cupboard first, because both abstract actions are enabled in the initial belief state.

4.2.2 Enhancing Optimality

The focus of the existing *BHPN* has been on correctness and computational efficiency often at the expense of optimality. In this section we introduce new mechanisms aimed at enhancing optimality with limited impact on efficiency, making the system applicable to the NAMO domain where optimized behavior is crucial to minimize unnecessary environment modifications.

4.2.2.1 Reconsideration

As we have seen, after an action is executed and the belief state is updated, it may be that the next step in an existing plan is not a good choice: it might be impossible to execute, or there might be better alternative courses of action. Replanning from scratch after every primitive operation and belief update will ensure that action choices are as up-to-date as possible. Yet, this would also be very expensive. In this section, we consider a series of approaches that are intermediate between attempting to execute an existing plan as much as possible and replanning at every opportunity. These methods fall into two categories: flexible execution of plans that we have already made, and selective replanning. We have already seen one form of flexible plan execution implemented in the current *BHPN*; we now present some novel extensions.

Satisfaction of existing goals We can also take advantage of *serendipity:* even though the particular subgoal, γ, that we are trying to achieve is not yet true, it might be that some subgoal in a more abstract plan (deeper in the stack) has been satisfied as a result of an unexpected action outcome or by nature. We can extend the idea of executing the rightmost executable operation in the plan at the top of the stack, and instead find the deepest (most abstract) plan in the stack that has a goal currently being expanded that is different from the rightmost goal (the final goal of the plan) at the next (more concrete) level of the stack. If there is such a plan, then we should begin execution from the next applicable step in that plan, instead. Lines 1 through 4 in Procedure 8 provide pseudocode for doing this, under the assumption that we are using the definitions in Procedure 6 and Procedure 7 as well.

Suppose that after carefully examining object A in the last part of Plan 4 in our example, the robot was to find that there are already two objects in the cupboard, then it would notice that the overall goal has already been achieved and return directly, rather than continuing to put A and then B in the cupboard. This is computationally inexpensive, with running time on the order of the total number of steps in all the plans on the plan stack, assuming that

Procedure 8: Reconsider

```
Input: $ps, b$
for $i$ in 0..len(ps) do
    $(\omega', \gamma') = ps_i.nextStep(b)$
    if $\gamma' \neq \mathrm{ps}_{i+1}.g_{\mathrm{ps}_{i+1}.n}$ then
        $ps.pop(i..len(ps))$
    if $p$.triggerReconsideration$(b)$ then
        $p' =$ Plan$(p.g_{p.n}, b, p.\alpha)$
        if $p' \not\subseteq p$ then
            $ps.pop(i..len(ps))$
            $ps.push(p')$
```

computing *nextStep* for a given plan requires examining each of its pre-images.

Selective replanning We might also wish to increase the robot's degree of responsiveness to change in its beliefs, while still trying to avoid unnecessary re-computation. In Procedure 8, lines 5 through 9, we potentially reconsider each plan in the stack.

We therefore allow plan instances to have an optional *trigger* method: $triggerReconsideration(p, b)$. This domain-dependent function decides whether it might be advantageous to re-generate a plan for the goal that p was intended to achieve. Such triggers will be executed at every level after every action, and so care must be taken to make them computationally light-weight. They are intended to return **true** in belief states for which it is likely to be worthwhile to change plans.

If replanning is triggered, then a new plan p' is created, with the same goal and abstraction levels as for plan p, but starting from belief state b. If p' is the same as p, or a substring of p (which is a natural occurrence, since some operations from p will, in general, have already been executed), then we go on to consider the next level of the plan stack. However, if p' is different from p, then the rest of the plan stack is popped off, p' is pushed onto it, and the planning and execution process resumes.

We have experimented with two types of triggers. The first is detection of major perceptual events, such as newly detected objects, that overlap with parts of the environment that are relevant to p. The second is re-evaluation of existing plans in the new belief state, and detection of a substantial increase in cost. Although it takes time exponential in the plan length, in general, to find a plan, it is only linear time to recompute its cost starting in a new state. If we cache the cost of the plan in the belief state for which it was constructed, and find that it has become more expensive due to unexpected changes in the belief, it might warrant replanning to see if there is a better alternative.

To avoid extraneous computation, we might create a cascade, in which primitive perceptual events may trigger re-evaluation, which may trigger replanning, which may trigger the abandonment of the remaining planning hierarchy and adoption of a new plan. In our example, in plan 4, the robot might find that in order to clear the swept volume for picking object A, it would have to move several additional (previously unseen) objects out of the way. That information could cause the cost estimate for action of picking A in plan 2 and even for placing A in plan 1 to be higher than they were initially, and trigger replanning that might result in the use of a different grasp for A. In fact it could cause the planner to choose an entirely different object to put into the cupboard instead of A.

If we trigger replanning only when the cost of an existing plan increases, the robot will not be able to be *opportunistic*, in the sense of taking advantage of new opportunities that present themselves. For instance, it may be that the use of an object, C, was initially discounted because it was believed to be very difficult to pick and place, because it was in a different part of the room, or highly occluded by other objects. If, during the course of looking for object A, the robot were to notice that C was actually easily reachable, it might be desirable to use C instead of A in the plan. In general, detecting that a plan involving C might be better requires as much time as replanning so one can, instead, always replan after every action.

4.2.2.2 Foresight

We can both reduce the replanning cost and the operational cost of taking extra actions by allowing our future intentions, as encoded in the plan stack, and our future observations, as encoded in the belief state, to inform choices we make in the present. We now discuss two new strategies that we have incorporated into *BHPN*.

Constraints from Intentions The planning problems solved by *Plan* occur in the space of probability distributions over states of the robot's world. In general, the world-state space is itself effectively infinite (or, at least, unbounded finite) dimensionality, with mixed continuous and discrete dimensions. There are many ways of grasping an object or picking a specific geometric goal configuration within a target region. The planner therefore requires methods of selecting concrete variable bindings. The use of *generators* in *BHPN* is described in detail in [40]. The generators use information in the current belief state and goal to select one or more reasonable values for variables. However, this process can be myopic, potentially making choices that make it more difficult to carry out future parts of the plan. To make better choices, the planner should be aware of constraints imposed by the future parts of the plans at every level of abstraction.

This is straightforward to accomplish, structurally. We can simply pass the entire plan stack into the call to *Plan*, replacing line 10 in Algorithm 3 with

$$ps.push(Plan(\gamma', b, p.\alpha.incr(\omega), ps)) \ .$$

For example, when clearing out the swept volume for picking up *A*, it might be necessary to place some objects out of the way. The intention to place *A* and then *B* into the cupboard could be taken into account when deciding where to put these other objects, so that they won't interfere with the operations of the rest of the plan[3].

[3]It is important to note that while this reasoning improves plan optimality, it is not required for correctness. During the time *BHPN* is performing the operation of placing *A* in the cupboard, it will, if necessary, move out any objects that are in the way, whether or not they were placed in the earlier part of the execution.

Predicted Observations Another form of foresight is to avoid reconsideration or replanning in the future by taking the robot's uncertainty into account when performing geometric planning. For example, knowing that when the robot gets close to an object and looks at it, the resulting pose estimate will be drawn from the current uncertainty distribution, the planner should select paths that are less likely to be obstructed when further information is gathered[4]. This strategy can improve both efficiency and optimality: efficiency because replanning is reduced and optimality because the actions are more likely, in expectation, to be appropriate for the situation.

The mechanisms of reconsideration and foresight described in this section are applicable to most robot domains, but their particular instantiation will depend on each domain. Similarly, their effectiveness will vary among domains. In the following section, we explore the instantion and effectiveness of our mechanisms for the NAMO domain.

4.2.3 Navigation Among Uncertain Movable Obstacles

The basic *BHPN* mechanisms that had previously been used in a pick-and-place domain were relatively straightforward to apply in the NAMO domain. We only had to make minor changes in the context of moving very large objects such as adjusting the path planning and object placement procedures to better model the arm and the object being held. In this section, we describe the particular methods that were used to implement foresight and reconsideration in NAMO.

4.2.3.1 Implementation on PR2

We address NAMO as a mobile-manipulation domain in which a real PR2 robot must manipulate chairs in order to clear its path to reach a goal, as shown in Fig. 20. The robot is given a basic map of the walls and permanent physical features of the domain, including

[4]This can be achieved by utilizing the previously introduced ε-shadows [40]. The ε-shadow of an object is defined to be the volume of space that with probability greater than $1-\varepsilon$ contains all parts of the object. These ε-shadows can now be treated as soft constraints for a geometric path planner. A heuristic cost penalty as a function of ε can be applied for each initial intersection with a shadow, biasing the planner to generate plans more likely to not intersect objects.

some objects that it can use as landmarks for localization, but there are obstructing chairs that it is unaware of when it begins acting. It uses a object-recognition system similar to the one discussed in Section 4.1 operating on RGBD data from a Kinect sensor to detect tables and chairs in the environment.

Because the robot is real, we must take uncertainty in actions and observations seriously. We use an unscented Kalman filter to estimate relative poses among the robot and other objects in the world as well as a volumetric representation of the space that has not yet been observed [40]. The planning system will ensure that any region of space has been observed before the robot moves into it, that an object is well localized with respect to the robot before it attempts to pick it up, and that the robot is reasonably well localized with respect to the map coordinate frame before it moves the base.

The simulation results included in this chapter are for a realistic simulation of the PR2, including simulated point-cloud perception and control of all degrees of freedom of the robot. It includes grasp planning, motion planning with an RRT, and view planning: the same code controls both the robot and the simulator.

4.2.3.2 The value of reconsideration

Exploiting serendipity is the conceptually simplest optimization mechanism, and does not require any domain-dependent specification. It occurs most frequently in the NAMO domain when the robot performs a *look* action to determine whether some region of space is clear. In so doing, it may discover that other regions it has planned to observe in the future have already been observed in the process.

The selective replanning mechanism requires domain dependent procedures. In the NAMO domain, we constructed the basic reconsideration trigger as follows:

1. Trigger the first stage of reconsideration if any new objects were detected that overlap any paths or placements currently being considered.

2. For any path that is under reconsideration, plan a path from the start to the goal,

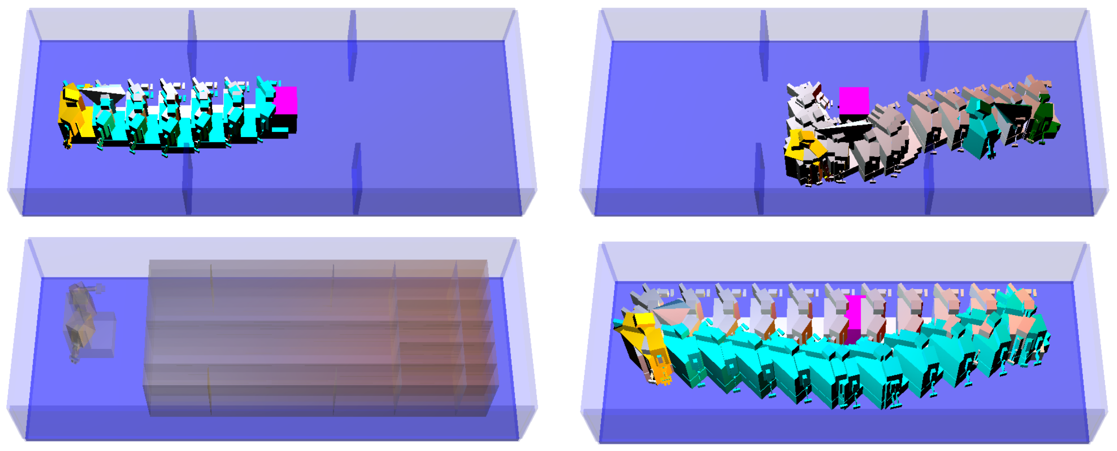


Figure 22: Serendipity example (top row); Reconsideration example (bottom row)

> using new information about the domain (including known obstacles and known free space); if the new path costs less than the original path (by some fixed amount to avoid needless switching) then trigger symbolic replanning. Also, for any object placement that overlaps a new object, trigger symbolic replanning.

We also experimented with conditions in which each of the stages of reconsideration were triggered after every action: this is more computationally costly, but offers the ability to detect new opportunities that may have arisen to increase optimality.

Fig. 22 shows a very simple example of reconsideration. The robot's goal is to move to the right end of the volume. As shown in the first frame, it has not yet observed most of the environment, so it is shrouded in *fog*. It makes a plan to move straight through to the end of the room. Then, as it looks to clear the fog, it finds an obstacle in its path. The original path and obstacle are shown in gray and magenta in the second frame. The appearance of a new object triggers reconsideration of the plan. Re-evaluation of moving along the original path reveals a much higher cost since the obstacle must be moved out of the way. Hence, replanning is triggered, resulting in a new path, shown in cyan, that avoids having to move the obstacle. In contrast, the original *BHPN* algorithm would have proceeded by having the robot move the obstacle out of the initially computed swept volume.

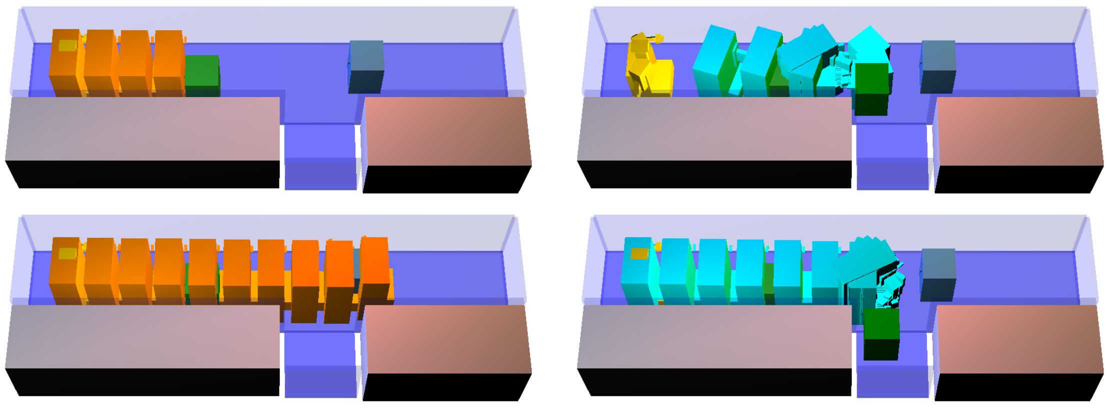


Figure 23: Behavior with and without considering future goals when selecting placements of obstacles.

4.2.3.3 The value of foresight

We implemented two foresight mechanisms. The first is motivated by the insight that, in the NAMO domain, the only decisions that can make future action significantly more costly are decisions about where to place objects. Those should be made with as much information about future intentions as possible. So, the generator procedures that make geometric choices about where to place objects when moving them out of the way examine the entire plan stack, and extract all instances of predicates that constrain some region of space to be clear. The generator tries, if possible, to find placements of objects that do not intersect any of these *taboo* placement regions.

Figure 23 illustrates a domain in which there are two obstacles, and the robot must reach the end of the hallway. It decomposes the problem into two parts: clearing the left side of the path to the goal, and clearing the right side of the path to the goal. In the first row, we see the behavior without foresight: the orange shapes are approximate robot volumes indicating the swept region to be cleared. It is determined that the green box must be moved out of that volume, and so a pose just past that region is selected for the box, as shown in the second image of the first row, and the robot places the box there. While satisfying the current problem of clearing the left side of the path, this placement is little helpful in achieving the robots overall goal of reaching the end of the hallway. *BHPN* does recover:

when it is time to clear the right part of the swept volume, both objects are successfully moved out and the goal is achieved. However, such a recovery carries the cost of moving the green box twice. In the second row, we see the same choice being made with foresight: the entire swept volume of the robot is treated as a taboo and, hence, the planner selects the *closet* at the bottom as a place for the obstacle, thus avoiding any unnecessary object motions.

BHPN has an existing mechanism for maintaining free-space connectivity that can also be seen as an example of this more general form of foresight. It operates under the assumption that the robot will always be required to move again, after the current sub-plan is finished, and so attempts to keep it as free as possible to continue. When selecting the placement of an object, the generator ensures that the following paths exist: from the robot's current configuration to the configuration for picking the object, with the hand empty; from the pick configuration to the place configuration, while carrying the object; and from the place configuration back to the initial configuration, with the hand empty, *and with the object placed in the target location.* These three conditions ensure that the pick and place operations can be planned in detail and that, after placing the object, the robot will be in the same connected component of free space as it was when it started. In the current implementation, this constraint is always applied; however, there are situations in which it is important to be able to violate it (e.g., when closing a door behind you) and it will be a matter for future work to use foresight mechanisms to determine whether or not to enforce connectivity.

Figure 24 demonstrates this reasoning: there is a relatively small swept volume from the robot to its goal (shown in the second frame of the top row), but there is an obstacle in the way. It is a simple matter to move the object out of the swept volume by dragging to the left, into the hallway. However, at that point, the robot would still not be able to reach the goal. Thus, a placement for the object is generated in the bottom left of the domain, ensuring that the robot stays connected to its original pose, and is then easily able to reach

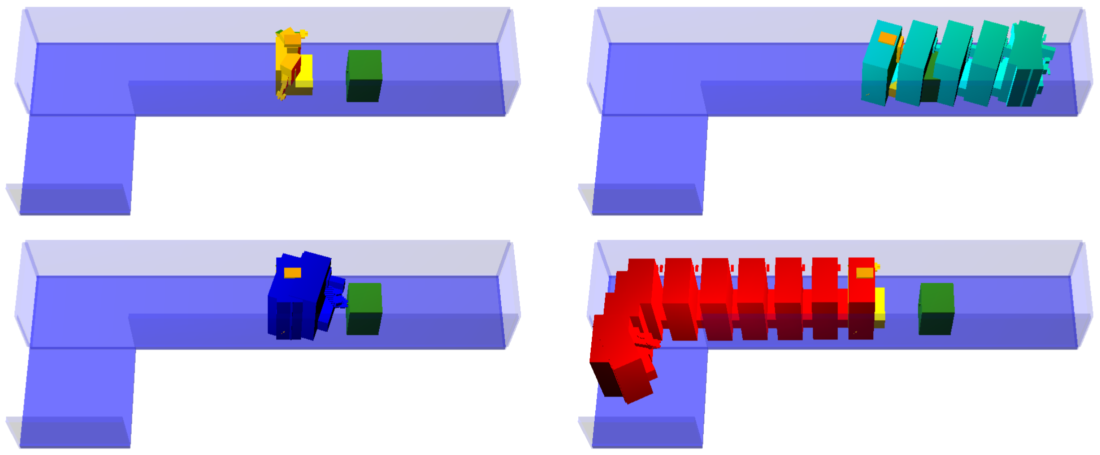


Figure 24: Maintaining connectivity of the free space during pick and place.

the goal. In the second row, the blue swept volume is from the robot's configuration to the pick configuration; the red volume is from the place configuration back to the original configuration.

Fig. 25 shows the robot using foresight to take potential future observations into account. It does so by using the current belief-state distribution on the poses of obstacles to construct "probability contours," which represent locations that might possibly be obstacles, depending on which observations are perceived. The robot needs to reach the right side of the domain, and can choose one of two hallways. Each contains an obstacle, but the pose uncertainty (shown as a lighter-colored "shadow" around the object) with respect to the obstacle in the top hallway is larger. The robot determines that it has a better chance of getting through without having to move an obstacle if it goes through the lower hallway, even though the geometric distance is longer, and plans the path shown in the image on the right. Once it approaches and achieves a better estimate of the object location, the robot finds that it can actually take a shorter path; the bottom frame shows the original path in gray and the new one in green. In order to do this last optimization, the robot must be triggering full reconsideration after a new estimate of object location, and not relying on an increase in the estimated cost of the existing plan to trigger replanning. This type of optimization has been used in previous work; we mention it here to illustrate the ease of

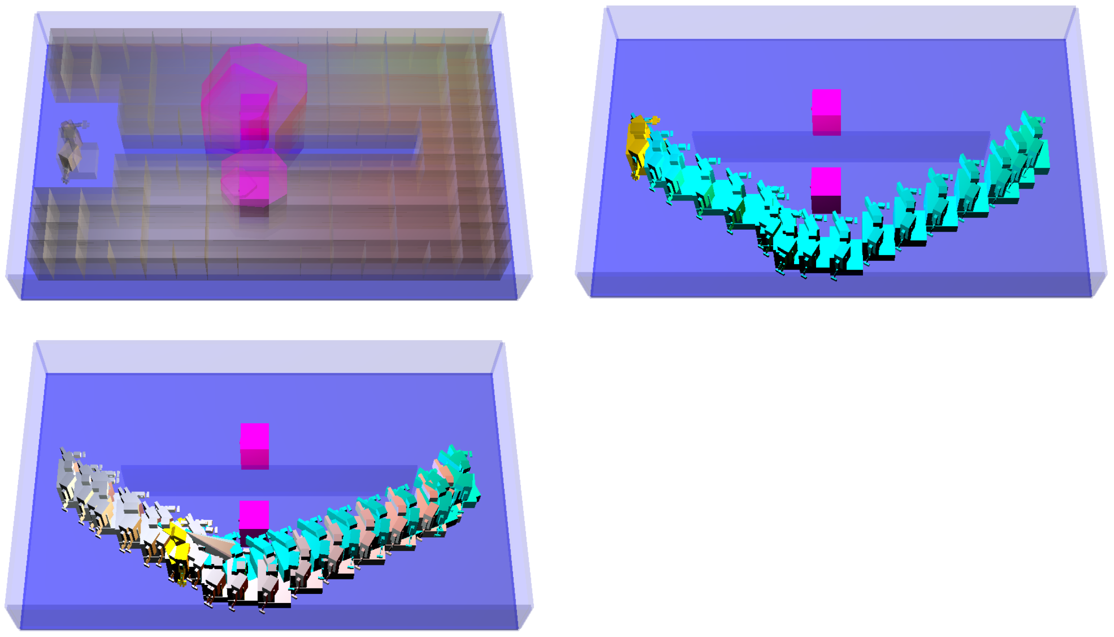


Figure 25: Using possible future observations to inform the choice of plans.

integrating such improvements in our framework.

4.2.4 Experiments

We compared three different versions of *BHPN* on different simulated domains, illustrated in Fig. 26. The domains have varying number of objects, object locations as well as start and goal configurations. All the algorithms include the foresight mechanisms described earlier; they differ in their approach to reconsideration.

- **Original *BHPN***: does not do any reconsideration, only replanning on failure.
- **Aggressive replanning**: replans after every action.
- **Selective replanning**: triggers replanning when new objects are discovered and when a new path to the goal is shorter than the existing plan by one meter.

For each algorithm in each domain, we measured the total simulated execution time (including all planning, simulated perception, etc.). The time column is the ratio relative to the baseline (original BHPN) times: domain 1 = 1265s, domain 2 = 1020s, domain 3 =

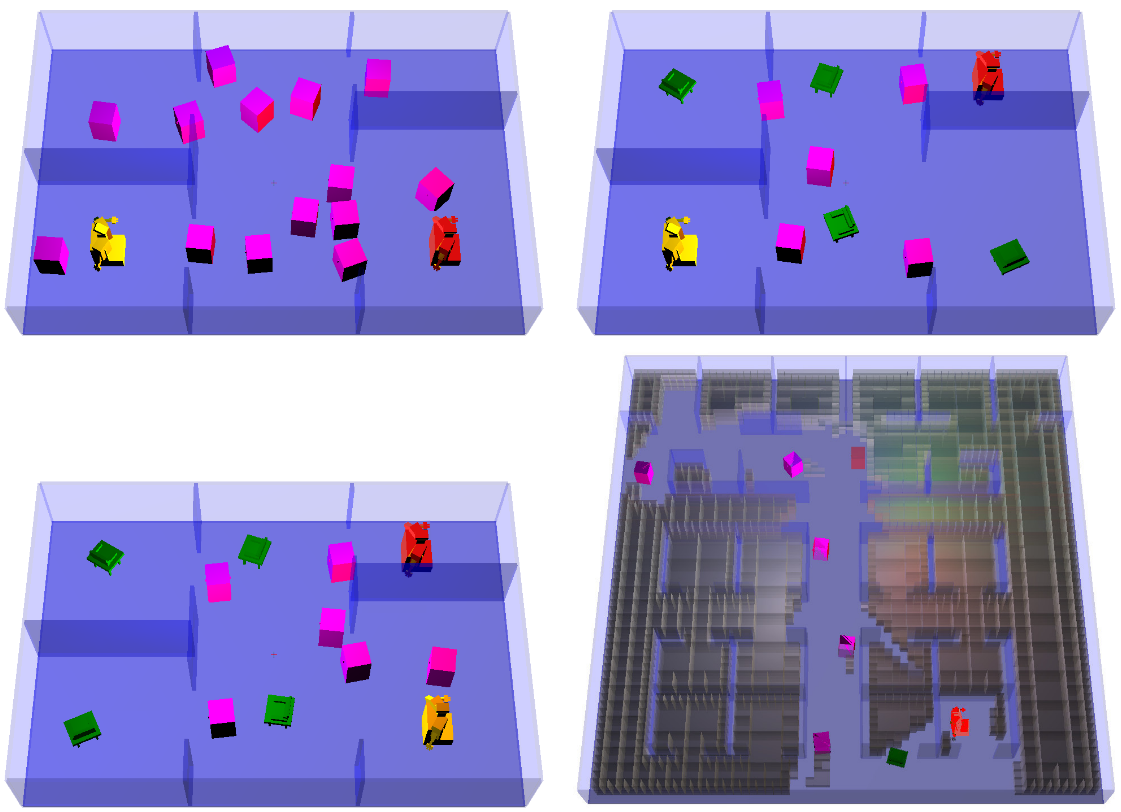


Figure 26: Domains: The first three are used in the experiments; the robot start and goal are in orange and red, respectively. The fourth domain was used to test scalability. In all cases the robot has no initial knowledge of the objects in the environment; the gray areas in the last domain show areas that were never explored.

Table 3: Results based on 137 runs. Best results in bold.

	Original *BHPN*				Aggressive Replanning			
Domain	picks/places	looks	moves	time	picks/places	looks	moves	time
1	4.75	4.25	12	1	2.1	3.2	7.8	1.01
2	3	**5**	11.8	**1**	**2**	6.3	11.1	1.66
3	3	6	10.3	1	**1**	5.8	8	1.40
Avg	3.6	5.0	11.4	1	**1.7**	5.1	9	1.33

	Selective Replanning			
Domain	picks/places	looks	moves	time
1	**2**	**3**	**6.8**	**0.57**
2	**2**	6.3	**10.7**	1.06
3	**1**	**5**	**7**	**0.89**
Avg	**1.7**	**4.8**	**8.1**	**0.81**

850s, average = 1045s. While these times also capture computations necessary for primitive action executions such as RRT calls for move actions, they do not include the time it takes to physically execute these actions by the robot. Given that the physcial execution time is typically highly system and parameter depended we instead report the number of primitive action executions. For the NAMO domain these actions are pick, place, move, and look. The results are summarized in Table 3.

The results show that aggressive replanning substantially increases the runtime of our simulations compared to the original *BHPN*. This is due to the fact that the current plan is discarded after every action execution, independed of the outcome or obtained observations. However, as aggressive replanning always generates plans in the current belief state, it drastically reduces the number of actions that have to be executed by the robot.

The selective replanning approach attempts to merge the benefits of the original *BHPN* and aggressive replanning by avoiding intensive replannings as well as action executions. And indeed we observe that selective replanning improves execution optimality over the original *BHPN* and efficiency over aggressive replanning. The computational efficiency of selective replanning is also improved over the original *BHPN* as less computation for primitive actions is necessary. These results support the concepts introduced in this chapter.

To verify the applicability to very large domains we modeled our office space in simulation (Fig. 26). The domain covers $100m^2$ and contains 50 objects. The robot had to navigate from the top left cubicle to the bottom right cubicle with prior knowledge of only the permanent obstacles. The robot was successfully able to reach its goal configuration while only requiring two object displacements. The right most visualization in Fig. 26 shows the robots final belief state.

We also ran experiments on a real PR2, shown in Fig. 20, to demonstrate the feasibility of this approach on a real robot. Again, the robot had no prior knowledge of the movable obstacles. Additionally it had substantial motion and perception uncertainty. The underlying *BHPN* mechanisms for robust planning and execution under uncertainty integrate effectively with reconsideration. Frequently using look actions at known landmarks to reduce positional uncertainty, the robot was successfully able to reach a goal configuration outside its current free-space region by manipulating two objects.

4.3 Discussion

In this chapter we presented a vision system allowing a robot to detect even partially occluded objects in the environment, and building upon these findings, described a hierarchical planning and execution architecture that incorporates concepts frequently used by humans: Reconsideration and Foresight. We showed that these concepts can be used to improve both efficiency and optimality for integrated task and motion planning and execution in large and challenging domains. The architecture enabled a real PR2 robot to solve NAMO domains with state, action and sensing uncertainty.

These results present a major step towards reliable, real-world robotic systems that are capable of autonomously moving objects out of the way. However, as discussed in Chapter 1, it is not always sufficient to just reason about moving objects out of the way. Instead, robots should also be able to reason about *using* environment objects. The next chapter therefore introduces the Navigation Using Manipulable Objects (NUMO) domain as a generalization of the NAMO domain that allows the robot to also consider using environment objects as tools.

CHAPTER V

NAVIGATION USING MANIPULABLE OBJECTS

In the previous chapters, we discussed methods that allow robots to autonomously move objects out of the way. While such capabilities are crucial for autonomous robotic systems deployed in a wide range of environments, ranging from typical households to disaster areas, it may not always be sufficient to just reason about moving objects out of the way. Consider grabbing a box and using it to get up on a higher platform or placing a board over a gap in order to step over it. While such reasoning patterns can make the difference between task completion and task failure, they are outside the scope of the NAMO domain.

We therefore now generalize the Navigation Among Movable Obstacles (NAMO) domain to the Navigation Using Manipulable Objects (NUMO) domain. In the NUMO domain, the robot is not restricted to just the moving of environment objects to clear a path, but it can also *use* them as tools to create a path in the first place. While this conceptually increases the solution space for the robot, the general complexity results obtained in Section 1.2 hold for the NUMO domain just as well as for the basic NAMO domain. In the NUMO domain the robot could use any manipulable environment object and reposition it to any configuration, again yielding exponential search space complexity. In fact, as discussed below, many of the methods developed for making the NAMO domain tractable, such as constraint relaxation [92, 93] and constraint back-propagation [95], can be adopted to the NUMO domain.

We now present two realizations of the NUMO domain. First, we present a system that allows a humanoid robot to overcome its inherit shortcomings of limited step width and height by using environment objects to change the environment geometry to its advantage. The system is applicable to scenarios in which a single object is sufficient to overcome an obstruction. We then present a second NUMO system that allows a robot to use existing knowledge about simple structures to also reason about ways of increasing the force it can assert onto the environment by gaining a mechanical advantage using environment objects.

This chapter assumes full world knowledge, including the position of the robot and all objects as well as task-related object attributes such as weight, support weight and grasp and drop primitives. This assumption is loosened in the next chapter.

5.1 Overcoming Geometric Constraints

This section presents a system that allows a humanoid robot to reason about changing the geometry of its environment to its advantage[1]. The following systems solves I_1UU problems (see Section 1.2).

Legged robots have unique capabilities to traverse complex environments by stepping over and onto objects. Many footstep planners have been developed to take advantage of these capabilities. However, legged robots also have inherent constraints such as a maximum step height and distance. These constraints typically limit their reachable space, independent of footstep planning. In this section we present a system that enabled a real robot to use a box to create itself a stair step or place a board on the ground to cross a gap, allowing it to reach its otherwise unreachable goal configuration. Fig. 27 shows an example execution of a HRP-2 robot using the proposed system.

In order to handle the exponential search space complexity resulting from considering environment modifications (Section 1.2), our proposed system combines existing work in footstep planning with the concept of constraint relaxation. The footstep planner is allowed to violate robot constraints such as maximum step width or height if necessary. While, the resulting path might not be directly executable by a locomotion controller, similar to usages within the NAMO domain, it provides a heuristic for determining how the robot should modify its environment to accomplish the task. For example, in Fig. 27 the initial footstep

[1]This work appears in [58].

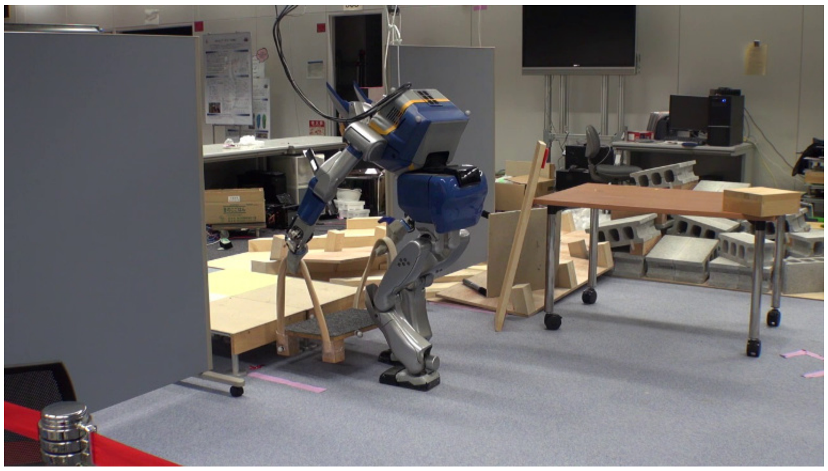


Figure 27: Robot decides to place a box in front of the platform to be able to step up the platform.

plan contains a constraint violation at the edge of the platform. The plan requires the robot to step higher than it physically can. The proposed system uses this information to guide the robot to grab the box and place it in front of the platform.

We now describe our method in more detail.

5.1.1 Algorithm

The algorithm takes advantage of existing research in humanoid locomotion planning and adopts the common approach of separating footstep planning from the locomotion controller [16]. However, to allow the robot to use environment objects as tools these methods have to be extended.

Considering all possible environment configurations is exponentially complex in the number of objects (Section 1.2). In order to resolve this complexity we apply the concept of constraint relaxed planning, which has previously been used to make the NAMO domain tractable [94, 95, 97]. For the NUMO domain, we allow the planner to violate the constraints of the robot in order to guide the decision making towards useful environment configurations. By creating a plan that violates the constraints, the planner can hypothesize a scenario where the robot would be able to follow a footstep plan to the goal, if only the environment was locally modified. This allows the planner to focus on these local modifications. To prevent the planner from unnecessarily violating robot constraints, a heuristic cost penalty is applied for each constraint violation. The magnitude of this penalty determines the planner's willingness to explore detours before considering environment modifications.

Inevitable inaccuracies in actuation that occur on a real robot system make it difficult to exactly execute a detailed long horizon plan [52]. If the robot is planning detailed motions based on future environment configurations, the robot risks executing actions not supported by the environment. We therefore interleave planning and execution in our system (Chapter 4). Later modifications are not planned for until after the robot completed its current modification and the actual resulting environment configuration is determined[2].

Algorithm 9 summarizes the proposed ETAP framework. The algorithm begins with calling a constraint relaxed humanoid locomotion planner (line 3). In case the resulting plan violates constraints, an object and a target location to resolve the first violation are determined (line 4 - 6). The robot then moves to a grasp position, updates its configuration and grasps the object (line 9 - 14). Again updating the configurations the robot moves to the object target location and places the object (line 15 - 20). The robot repeats this procedure until it is at the goal.

5.1.2 Implementation

While the previous section provided a general overview of the proposed system, this section provides details of our implementation.

5.1.2.1 Constraint Relaxed Planner

The algorithm presented above requires a humanoid locomotion planner extended with the concept of constraint relaxation. We build on the planner presented in [16]. The planner performs an A* search through the space of possible footstep actions and locally adapts

[2]An alternative method is to trigger re-planning if too much divergence from the plan occurs. However we empirically found that this is too expensive in practice for the long horizon tasks considered in this work and requires substantial tuning of the re-planning thresholds.

Algorithm 9: NUMO for Geometry Constraints

```
Input: $R$: robot, $\mathscr{S}$: static objs, $\mathscr{O}$: manipl. objs, $g$: goal
1  while R not at goal do
2      UPDATE(); // update robot and obj loc
3      moPl ← CONSTR_RELAXED_PLANNER($R, g$);
4      if moPl contains constraint violations then
5          $v$ ← GET_FIRST_VIOLATION(moPl);
           // get obj and obj target:
6          $(o, o\_t)$ ← RESOLVE_VIOLATION($v$, $R$, $O$);
7          if o is NULL then
8              return; // goal not reachable
9          $o\_g$ ← GET_GRASP_POS($R, o$);
10         pickPl ← LOCOMOTION_PLANNER($R, o\_g$);
11         R.execute(pickPl);
12         UPDATE();
13         grasp ← GET_GRASP_MOTION($R, o$);
14         R.execute(grasp);
15         UPDATE();
16         dropPl ← LOCOMOTION_PLANNER($R, o\_t$);
17         R.execute(dropPl);
18         UPDATE();
19         drop ← GET_DROP_MOTION($R, o, o\_t$);
20         R.execute(drop);
21     else
22         R.execute(moPl);
```

the action set if some actions are not possible due to environment properties. To allow the planner to violate robot constraints if necessary, we extend the action set the planner uses. We add actions for stepping substantially higher and further than the actual robot can do. To avoid unnecessary environment modifications, these actions are assigned a high cost. The increased cost of these actions leads the footstep planner to first explore longer environment paths before considering paths that include constraint violating actions.

5.1.2.2 Constraint Resolution

To resolve a constraint violation, an appropriate environment object and target location need to be chosen. In analogy to affordance [28], and similar to [27], we include a task-related object attribute representation. In our implementation, each object contains the following attributes:

- (x, y, z, θ): position and orientation,
- *weight*: the object's weight,
- *maxW*: the max weight the object can support,
- $\mathscr{FS}$: a set of flat surfaces the robot could potentially step on defined relative to the object center,
- $\mathscr{S}$: a set of surfaces defined relative to the object center that have to be supported by flat ground,
- *maxD*: the max height difference that is allowed between the surfaces in $\mathscr{S}$,
- $\mathscr{GC}$: a set of possible grasp configurations for the object defined relative to the object center,
- $\mathscr{G}$: a set of grasp primitives for the object
- $\mathscr{D}$: a set of drop primitives for the object

For a given constraint violation, the algorithm first selects all manipulable objects that could potentially be used to modify the environment such that the robot could traverse the according location. In a pre-processing step the algorithm rejects all objects that do not fulfill the minimum requirements to support the robot. This includes support weight and the size of the biggest surfaces in $\mathscr{FS}$ as well as all objects that are too heavy for the robot to manipulate. The remaining objects are then further filtered according to the specific violation at hand. For example, if the robot is required to step higher than it can, only objects whose height is such that the robot can step on it and can step the remaining height are preserved. Similarly, if the violation required the robot to step further than supported, the algorithm only keeps objects whose largest side of any surface in $\mathscr{F}\mathscr{S}$ is at least as large as the required step width. All remaining candidate objects are then sorted based on distance to the current robot configuration.

The sorted list is then iterated through and the first object that can successfully be reached by the robot and placed at the constraint violation location is returned. To verify reachability of an object the algorithm iterates through $\mathscr{G}$ and calls a non-modified version of the footstep planner presented in [16][3]. In order to find a target location for the object, the planner utilized the 2.5D grid map used by the footstep planner. The planner performs a local search around the coordinates of the violation and the last valid action within the actual plan and attempts to place the object. A successful object placement is achieved if all surfaces defined in $\mathscr{S}$ are supported by flat ground with a height difference of less then *maxD*. The local search perimeter is set such that a placement at any location within the perimeter would still resolve the constraint violation. We typically set the perimeter size equal to a robot step distance. For example, this would allow the planner to place the object at most step distance from the platform in Fig. 27, still allowing the robot to step on the platform if on top of the box. If the object is either not reachable or no target location can be found, the next object in the list is checked[4]. This procedure is summarized in Algorithm 10.

[3]As we are only interested in verifying reachability at this point, we use a relatively large goal threshold here to reduce planning time.

[4]Our implementation does not verify reachability between the object grasp configuration and the drop location. We assume that the robot can at least return to its current location with the object and then move to the drop location. However, it is straight forward to extend the algorithm to also verify drop location reachability directly.

Algorithm 10: RESOLVE_VIOLATION

```
    Input: $v$: violation, $R$: robot, $O$: manipl. objs
    Output: $o$: suggested obj, $o\_t$: suggested target loc
1   $candidates \leftarrow \{o \in O | o$ can be used to resolve constraint$\}$;
2   SORT_BY_DIST($candidates$, $R$);
3   for $o \in candidates$ do
4       if $R$ cannot reach $o$ then
5           continue;// skip $o$
6       $o\_t \leftarrow$ FIND_TARGET_LOCATION($o$, $v$);
7       if $o\_t$ is NULL then
8           continue; // skip $o$
9       return ($o$, $o\_t$);
10  return (NULL, NULL); // Failure
```

5.1.2.3 Object Grasping and Dropping

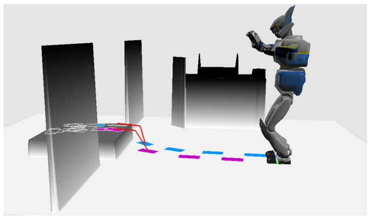
If the robot has reached the grasp configuration for an object it needs to grasp the object. Similarly if it has reached the drop location for an object it needs to drop the object. To achieve this the planner simply selects a motion primitive according to its configuration relative to the object or target location.

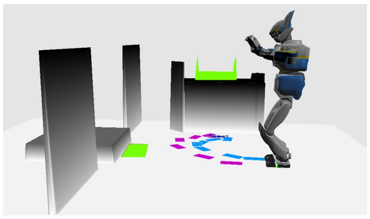
In our implementation the motion primitives are sets of precomputed motions defined as empirically sampled joint-space configurations. The joint angles are determined for different objects and grasp configurations. To execute a specific primitive, a spline-interpolated trajectory of the according joint angles is computed. The trajectories are tracked by PID control. The weight of the object is then incorporated or removed from the robot model respectively.

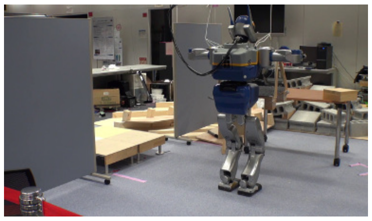
5.1.3 Experiments

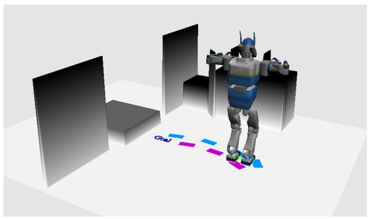
To verify the proposed algorithm, we implemented it on an actual robot system and in simulation.

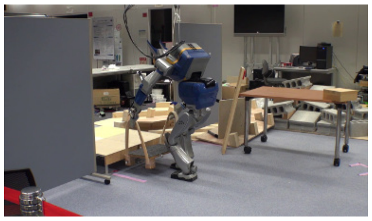
(a) Start configuration

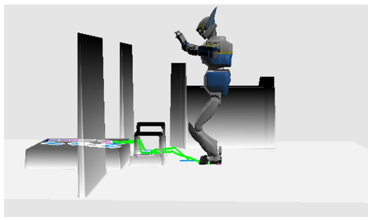

(b) Constraint relaxed planning output

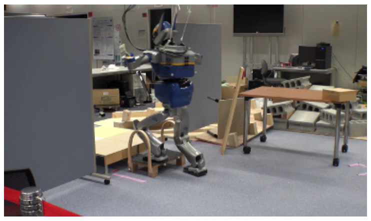

(c) Constraint resolution output


(d) Configuration after object pickup


(e) Drop planning output


(f) Drop motion execution


(g) Constraint relaxed planning output


(h) Locomotion to goal configuration

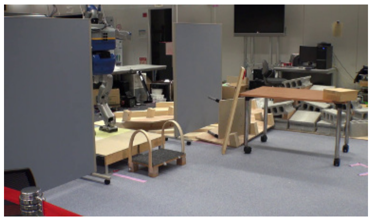

(i) Final configuration

Figure 28: Proposed humanoid locomotion planning system: Stairs Example.

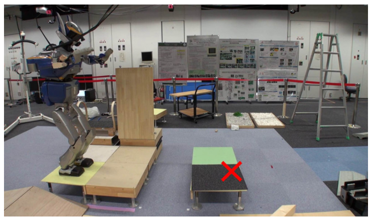
5.1.3.1 Real Robot

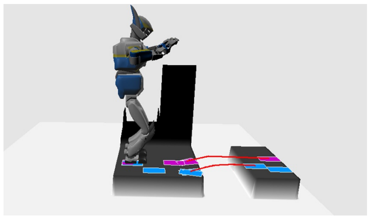
We implemented our algorithm on the HRP-2 robot platform and utilized existing locomotion and balance controllers for the robot [73] in our experiments. We also used a real-time motion capture system [96] to localize the robot and all objects within the $25m^2$ workspace.

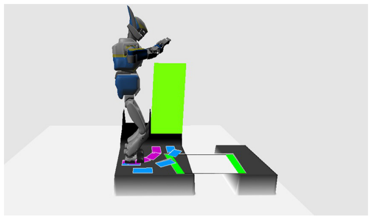
We tested our system in different scenarios.

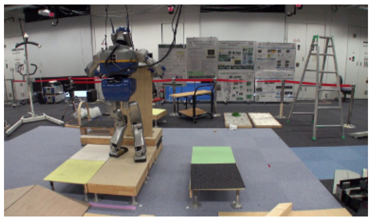
Stairs The first scenario can be seen in Fig. 28. In this setup the robot was commanded to take on a configuration at the end of the platform in front of it, as visualized in Fig. 28(a). The platform was higher than the robot can step.

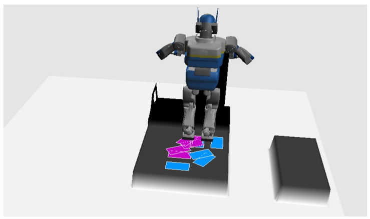
Fig. 28(b) visualizes the output from the constraint relaxed planning step (Algorithm 9,


(a) Starting configuration

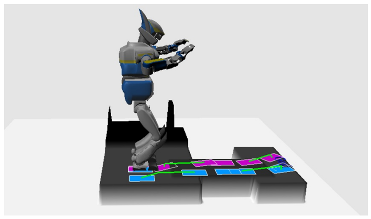

(b) Constraint relaxed planning output

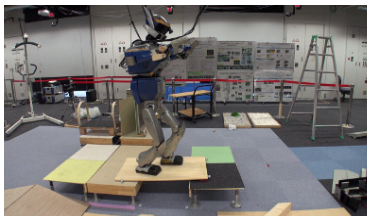

(c) Constraint resolution output


(d) Configuration after pickup


(e) Drop planning output

(f) Drop motion execution


(g) Constraint relaxed planning output


(h) Locomotion to goal configuration

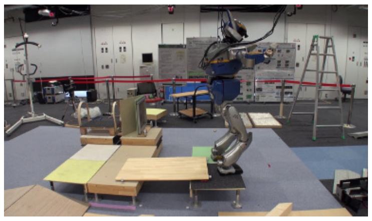

(i) Final configuration

Figure 29: Proposed humanoid locomotion planning system: Bridge Example.

line 3). The constraint relaxed planning step found a path to the goal, however it required the robot to step up higher than it physically can. The constraint resolution step (Algorithm 9, line 6) then determined an object to resolve the constrained violation at hand as well as a suitable target location, visualized in Fig. 28(c). The planner picked the box as it could support the robot and had the correct height to allow the robot to step on the box itself and consecutively on the platform. The entire bottom surface of the box was required to be placed on flat ground. The robot was then guided to pickup the selected object (Algorithm 9, line 9 - 14). Once the robot picked up the object (Fig. 28(d)) and updated its and the object's configurations, the robot was guided to the target location for the object and executed the appropriate drop motion primitive (Algorithm 9, line 15 - 20), visualized in Fig. 28(e) - 28(f). The algorithm then looped and tried again to find a path to the goal given the new environment configuration. Fig. 28(g) shows the output of the constraint relaxed planning step. This time a path to the goal was found without any constraint violations and the system guided the robot to the goal configuration without any further environment modifications (Fig. 28(h) - 28(i)).

Bridge Fig. 29 shows a second execution example. In this setup the robot was tasked with reaching the other platform. The gap between the platforms was too wide for the robot to step over. The constraint relaxed planning steps determined that the robot needs to cross the gap in order to reach the goal. The planning system then guided the robot to pick up the board and drop it across both platforms. The planner picked the board as it was longer than the gap. Additionally, the board only required 8cm on each side to be placed on flat ground if dropped. This allowed the robot to drop it across both platforms. After the robot verified that it could reach the goal given the new environment configuration, the robot stepped over the board and reached its goal configuration.

5.1.3.2 Simulation

We additionally performed simulated experiments over 10 different domains with varying sizes and difficulties. The domains contained between 3 and 18 objects and the robot typically needed to perform between 1 and 3 environment modifications to reach its goal configuration. Fig. 30 shows an example domain that required the robot to perform 2 environment modifications to reach its goal.

The average computation time for the constraint relaxed planning step was 30.36s when no path without constraint violations existed and 3.06s otherwise (typically after the robot modified the environment). We can see that if no direct path to the goal existed this step took roughly 10x as long as if a direct path existed. This is because we used a very high cost for actions that violate robot constraints to avoid unnecessary environment modifications. Consequently the planner explored a larger space before using the constraint violating actions. If no constraint violations were necessary, the planner could explore the space more

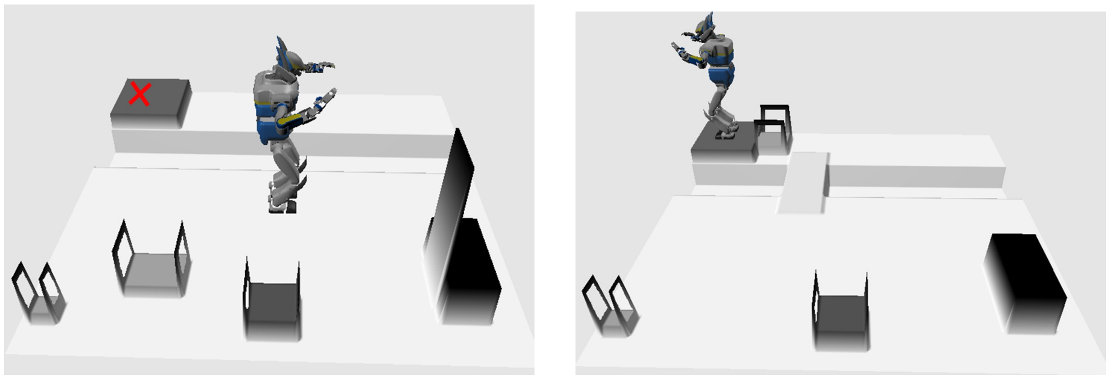

(a) Starting configuration (b) Final configuration

Figure 30: Simulation visualization. The robot is tasked with getting on top of the platform. The platform is higher than the robot can step. Additionally it cannot step over gap. The robot placed the board over the gap and used a box to step up the platform.

efficiently.

The average time for determining an object to resolve a constraint violation was 2.82s. The planning times to the grasp or drop location was 8.69s. Interestingly, the average planning times for a path to a pickup or drop location were almost 3x higher than for the constraint relaxed planning case if no constraint violations were necessary despite the planner behaving similar in these cases. This was caused by the fact that our implementation required a substantially higher goal configuration accuracy for pickup or drop location planning than for the final goal configuration.

The average execution time for the robot including pickup and drop motions was 84.52s, dominating the planning time.

5.1.4 Discussion

In this section we presented a planning system for the NUMO domain that is applicable to domains where a single object can be used to overcome an obstruction. The system was successfully executed on a real robot and enabled the robot to reach a goal configuration outside of its reachable space by building itself a stair step and a bridge. To the best of our knowledge this is the first time a humanoid robot showed such a behavior fully autonomously.

5.2 Overcoming Force Constraints

The previous section introduced a realization of NUMO for geometry constraints. We now present a NUMO realization for force constraints[5]. The following framework requires access to all relevant attributes of applicable physical structures and operates on I_kUU problems (see Section 1.2).

Just like humans, robots have a maximal force they can apply to the environment. Humans overcome their force limitations by utilizing knowledge about *mechanical advantage* afforded by possible combinations of objects and find creative solutions such as using a rod and an office cart to achieve leverage, or weighing down a cart with books to create a battering ram of sufficient mass and momentum. Robots should be able to show similar behavior. While a general and complete solution to automating these kind of reasoning patterns is beyond the state of the art, we argue that aiming for such abilities is a crucial research direction and present a first step into this direction as a realization of the NUMO domain.

Our approach is based on two significant properties. First, our system reasons backwards from the desired outcome to the force required to achieve it. This builds on semantic [31] and geometric [95] backward chaining, by adding dynamic goals and constraints. We achieve this by exploiting Newtonian mechanics, which relate force to object motion, to *solve for required forces*. This is an analytical solution which is more efficient than sampling forces and computing outcomes over a broad set of simulated trials. Second, we leverage prior knowledge about how objects might be combined to achieve useful effects. We formalize object combinations in terms of constraints on their relative configuration. This representation defines a mapping between a sub-space of possible object configurations and the outcomes which these configurations can be used to produce. Together, these define the necessary components for a regression-based planner in object-configuration

[5]This work appears in [61].

space. The first gives a precondition for reasoning from motion to force requirements, and the second provides a way of back-chaining from force requirements through a compact space of object configurations.

A baseline approach for regression planning in object-configuration space uses simple back-chaining. This means making sequential decisions without altering previous decisions except when no solution can be found. We argue that, although well-defined, this baseline approach is not sufficient because the choices at different levels of the plan fundamentally depend on each other.

To address this challenge, we present a method that reduces the search space by performing *constraint propagation* using relative configuration constraints across multiple levels of the plan. Our approach allows the planner to *jointly* consider multiple objects, and efficiently prune away incoherent configurations.

We demonstrate constraint propagation and baseline regression planning applied to the challenging problem of opening a jammed door. Our results show that simple back-chaining is too aggressive for multi-object placement problems, and cannot discover solutions which require decisions in different orders, such as moving a fulcrum into place before searching for lever configurations. In contrast, constraint propagation is capable of finding solutions even in cases where objects must be placed out of the order defined by the constraints, such as stacking books on a cart before ramming it into a door.

5.2.1 Problem Specification

We specify the environment as the following set of entities:

- R - a robot,
- $\mathscr{S}$ - a set of static objects,
- $\mathscr{O}$ - a set of movable objects,
- G - a goal object which must be moved.

The task is to determine a plan that allows the robot *R* to move the goal object *G* in some direction, even if this requires more force than the robot can exert on its own. We focus on two possible motions of a door, depending on its swing direction and state:

1. Jammed, partially opened doors that open inwards, as in Fig. 1(a) and

2. Locked, closed doors that open outwards, as in Fig. 1(b).

The planner is allowed to utilize any objects of the set *O* as well as *S*. As the previous section, this work assumes full world knowledge such as object locations and properties. Object properties are defined as physical attributes including weight, shape and affordances [28] such as rollable, graspable and stackable. We also assume that an estimate of the required force or impact momentum to open the door is available.

5.2.2 Approach

There are two critical components for a regression-planner to be capable of using objects in the environment to achieve goals. The first is a *physics model* which encodes rigid-body dynamics, and can be used to solve for the *forces required* to achieve *desired motions*. This requirement can be satisfied by an off-the-shelf rigid body physics engine. The second requirement is a compact representation which maps object configurations to the forces they can achieve. This issue is more challenging, and is the focus of the current work.

In this section we present a constraint propagation approach which allows the planner to restrict its attention to *coherent configurations* of objects. The core insight is that before searching over object configurations, constraints such as necessary mechanical advantage, required force transmission angle and fulcrum point height can be used to reduce the search space without eliminating potential solutions. The reduced search space can then efficiently be sampled.

To implement this idea for the jammed-door problem our algorithm utilizes two subroutines to, depending on the state of the door, determine either a lever or a battering ram mechanism. We develop a constraint representation for each mechanism and describe how they can be instantiated using arbitrary objects available in a scene.

5.2.3 Lever Like Structure

We first develop a constraint representation for lever-like structures, a common force-amplifying mechanism. We denote the maximal force the robot effector can apply as F_r and the minimal force needed to move the target object as F_t. The robot should consider the lever system for cases in which $F_r < F_t$. In this case the planner needs to determine a lever configuration that allows for a mechanical advantage such that the effective force on the target (*e.g.* door) is at least F_t.

A lever system defines a relation between four objects: the lever itself o_l, a fulcrum o_f, the payload o_p (*i.e.* the door), and the effort o_e (*i.e.* the robot effector). We then define the contact points for each pair of objects i, j of the lever system as cp_{ij}, which specifies a contact between object o_i and object o_j, in the coordinate-frame of object o_i (*e.g.* cp_{dl} denotes the point on the door at which the lever applies force). Where necessary we use a superscript w to denote the corresponding world-frame point.

Leverage constraints must define the space of legal poses for the fulcrum and lever which satisfy the following properties:

1. The lever end is in contact with the target payload (*i.e.* cp_{ld} and cp_{dl} exist and correspond to the same world point).

2. The effort point cp_{le} on the lever o_l is reachable by the robot effector o_e.

3. The fulcrum o_f is placed such that the attainable mechanical advantage given the robot's maximal force output F_r is sufficient for the required payload force F_t.

In addition, neither the robot nor the lever object should be in collision with the environment other than at the intended contact points.

Leverage constraints are then formalized by the following necessary system of equations:[6]

$$
\begin{aligned}
||cp_{ld} - cp_{lf}|| + ||cp_{lf} - cp_{le}|| &\leq length(o_l) && (18)\\
F_t &\leq \frac{F_r ||cp_{lf} - cp_{le}||}{||cp_{ld} - cp_{lf}||} && (19)\\
|\vec{F_D} \cdot [cp^w_{ld} \vec{-} cp^w_{lf}]| &\leq \varepsilon && (20)\\
cp^w_{el} &\in \mathscr{C}^{reach}_R && (21)
\end{aligned}
$$

Where $length(o_l)$ denotes the length of the major axis of object o_l, $\vec{F_D}$ the force direction necessary to open the door and $\mathscr{C}^{free}_R$ the configuration space region that the robot can reach. Eq. 18 states that the sum of the distances between the fulcrum point and contact points cannot exceed the length of the object. Eq. 19 adds an additional constraint on these distances ensuring that the minimum mechanical advantage needed to resolve the constraint is achieved. Eq. 20 requires that the angle between the desired force transmission direction and lever has to be close to 90°, and Eq. 21 ensures that the robot grasp point of the lever has to be within the same free space region as the robot.

Assuming one was given an assignment of objects, these equations can be used to construct the space of valid lever configurations in the world frame. We denote this space as the *leverage constraint space*. We can further subdivide this space into two sub-spaces:

1. The fulcrum constraint-space: a 3D volume in which the fulcrum point must lie such that all four conditions are satisfied for a given lever.

2. The lever constraint-space: a 2D plane defined by the fulcrum, the payload, and their respective normals (must be anti-parallel).

Note that these sub-spaces are interdependent: the set of valid fulcrum points depend on

[6]The lever object also has to be able to withstand the forces without breaking. While this can be computed using Euler-Bernoulli theory, we assume that objects do not break in this work.

the orientation of the lever which will be placed against it. This implies that, in principle, we must re-compute the fulcrum constraint space for every candidate assignment of the lever, and vice-versa.

We resolve this by placing additional constraints on the lever which permit us to solve for its orientation uniquely. These additional constraints are (a) that the lever must be horizontal, and (b) that the payload and action are at opposite extrema of the object. The first of these is chosen because it maximizes the applied force in the horizontal direction as required for opening the door[7], and the second is chosen because it maximizes the leverage effect for the given object. This allows us to focus on the fulcrum constraint sub-space.

The fulcrum constraint space is only well defined for a given assignment to the lever and fulcrum object. In order to be able to use the constraint space as a sampling space for determining objects and configurations simultaneously, we construct an upper bound constraint space that is guaranteed to contain the fulcrum. This is done by instantiating the constraint space construction with the least limiting objects.

Procedure 11 summarizes the proposed subroutine. To be able to enforce Eq. 21 the procedure begins by computing the reachable space of the robot, line 2. Further, to obtain an upper bound on the distance the fulcrum point has to have to the door, the procedure determines the length of the longest object in the environment that could be used as a lever. Given this length, Eq. 18 and Eq. 19 can be used to solve for the distance $||cp_{dl} - cp_{fl}||$, yielding the desired upper bound (line 3-4). To build up the fulcrum constraint space, the procedure now does a 360° sweep with the computed distance from the edge of the door and checks for valid configurations. This is done by verifying that the robot could reach the potential endpoint for a lever placed at that configuration (Eq. 21), there is no collision with the door itself and the force transmission angle is within ε (Eq. 20). The resulting constraint space is guaranteed to contain the fulcrum point if a solution exists. Fig. 31(a)

[7]The alternative involves placing the lever at an angle to the horizontal and either continuing to apply forces in the horizontal plane (sacrifices leverage), or applying forces along the new lever plane. This results in force components along the contact surfaces that may result in slippage.

Procedure 11: 1: Lever Structure

```
1  $cs \leftarrow \emptyset$;
2  $C_R^{reach} \leftarrow getReachableSpace(R)$;
3  $ln \leftarrow getMaxObjLength()$;
4  $maxFpDist \leftarrow \frac{F_r ln}{F_t + F_r}$;
5  $l \leftarrow bar(maxFpDist)$;
6  **while** *rotate $l$ at door edge* **do**
7      **if** *$cp_{el}^{w}$ would not be in $\mathscr{C}_R^{reach}$* ***or***
8      *$l$ in collision with door* ***or***
9      *force transmission angle difference within $\varepsilon$ of* $90^\circ$ **then**
10         continue;
11     $cs \leftarrow cs \cup \{$space covered by $l$ with door height$\}$;
12 $cs \leftarrow cs \setminus \{$space not coverable by edges$\}$;
13 **while** *not at threshold* **do**
14     $o_f \leftarrow getRandomFulcrumObj()$;
15     $edge \leftarrow$ get random edge on $o_f$;
16     $place(o_f, cs, edge)$;
17     $o_l \leftarrow getObjMinLen(edge, door)$;
18     $place(o_l, edge, door)$;
19     **if** *place succ* ***and*** *force direction correct* **then**
20         **return** placements;
21 **return** ()
```

visualizes an example output of this step.

Further, as the fulcrum point has to lie on an edge of an environment object, we can limit the height of the computed constraint space to the largest height of any potential environment object, reducing the constraint space even further. Fig. 31(b) shows the final constraint space for this example.

Given the constraint space, the procedure then proceeds by sampling valid parameter assignments. This is done by randomly selecting an edge of an environment object that could be used to create a fulcrum point[8] and placing it at a sampled configuration within the constraint space. Given this configuration, the procedure computes the minimal length an environment object needs to have for this particular configuration, randomly selects

[8]In our implementation these objects were determined based on their height relative to the robot workspace and the estimated slippage.

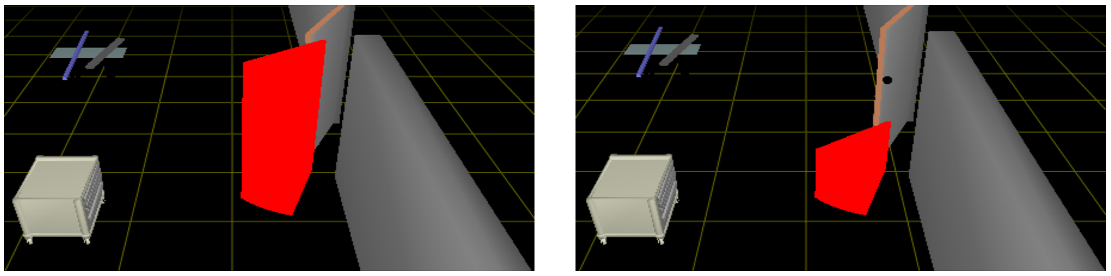

(a) Contraint space after 360° swipe. (b) Contraint space after potential fulcrum objects are considered.

Figure 31: Constraint space computation. The constraint space is visualized in red.

one of the possible objects and attempts to place it. This is repeated until either a valid configuration is determined or the user defined number of attempts is exceeded, line 13-20.

5.2.4 Battering Ram Structure

Next we consider the subroutine for determining a battering ram for cases in which the door is locked and opens outwards. The planner needs to determine which object to use as a ram, and where to place it such that it has a clear path to accelerate into the target object.

In this section we develop a constraint representation for a battering-ram using ground-supported (non-handheld) objects such as an office cart. The key concept that allows a battering ram to be a multi-object structure is that additional objects can be attached to a mobile body to increase its mass and momentum.

Overall, a battering ram system defines a relation between the target body o_t, a mobile body o_m to use as a ram, and zero or more auxiliary bodies $(o_i)_{i=0}^{k}$ which the robot can use to augment the mass of o_m. Executing a ram action involves accelerating o_m over a linear trajectory which terminates at the desired contact point cp_{tm} on the target body. To define the action we must therefore specify appropriate initial poses for each object in the system. We use ip_i^j to denote the initial pose of object o_i in the frame of o_j, and ip_i^w the corresponding world pose.

Ram constraints must define a space of valid initial poses ip_m^w for for the ram, and the local configurations $(ip_i^m)_{i=1}^k$ of all k auxiliary bodies which satisfy the following properties:

1. The total mass $M(o_m) + \sum_{i=1}^k M(o_i)$ is large enough to provide the necessary impact momentum for some impact velocity v_i.
2. The distance between o_m and o_t is large enough to attain v_i given the maximal applicable force F_r.
3. The swept volume of o_m along a linear trajectory between ip_m^w and cp_{tm}^w is collision-free.
4. All auxiliary objects o_a^i are statically supported by o_m.

Given an estimate of the required impact momentum p_t, we can then place mass and position constraints on the mobile body using the equations for momentum, work and kinetic energy:

$$p_t = mv \qquad \textit{(momentum)} \qquad (22)$$

$$F_r d = \frac{1}{2}mv^2 \qquad \textit{(work and kinetic energy)} \qquad (23)$$

with d denoting distance and m mass. This yields the following constraints:

$$d \geq \frac{p_t^2}{2mF_r} \qquad \textit{(distance constraint)} \qquad (24)$$

$$m \geq \frac{p_t^2}{2dF_r} \qquad \textit{(mass constraint)} \qquad (25)$$

Eq. 24 states the minimum distance needed for an object with mass m and a maximal robot force F_r. Eq. 25 on the other hand expresses the minimum weight an object has to have to achieve the necessary impact momentum given d and F_r.

As with the lever system, the battering-ram system contains interdependent constraints: the necessary mass of o_m depends on how much space the robot has to accelerate it before

Procedure 12: 2: Ram Structure

```
1  $max\_d \leftarrow$ getMaxDist(door);
2  $min\_m \leftarrow \frac{p_t^2}{2max\_dF_r}$;
3  for $o \in \mathscr{O}$ do
4      if $o.m > min\_m$ or $o$ not manipulatable then
5          continue;
6      $d \leftarrow \frac{p_t^2}{2o.mF_r}$;
7      try($o$, $d$, $door$); // check collisions etc.
8      if $succ$ then
9          return ($o$, $d$);
   // no solution found, attempt to increase weight:
10 for $o \in \mathscr{O}$ do
11     if $o$ not manipulatable then
12         continue;
13     $m^+ \leftarrow \frac{p_t^2}{2max\_dF_r} - o.m$;
14     attemptWeightIncrease($o, m^+$);
15     if $succ$ then
16         $d \leftarrow \frac{p_t^2}{2o.m^{new}F_r}$;
17         try($o$, $d$, $door$);
18         if $succ$ then
19             return ($o$, $d$, $addedObjs$);
20 return ()
```

it impacts the target object, and the necessary space depends on the (maximum) available mass of the mobile body. We address this by first assuming that the mass of an object is fixed and attempting to find a solution according to the constraints. If this is not possible, we reason about how much the mass has to be increased according to the maximal distance available.

Procedure 12 summarizes this subroutine. The function first determines the actual maximal distance available in the environment and then uses Eq. 25 to compute the minimum mass necessary to achieve the required impact momentum. Given this information, the procedure now iterates through the list of movable environment objects. For each object that is heavier than the required weight and the robot can manipulate, it is verified if it could be used (line 3-9). In our implementation this verification step included collision checks from the robot's current configuration to the object, from the object's configuration to its target location and finally for the swept volume necessary to accelerate the object. The first object that is found to work is returned. However, if no object is available in the environment that fulfills the weight constraint, this step cannot discover a solution.

In that case the mass constraint can be propagated forward and the procedure attempts to increase the weight of existing objects in the environment such that the necessary momentum can be achieved. This is summarized in line 9-19. The procedure starts by iterating through objects the robot could use and utilizes Eq. 25 to compute how much mass needs to be added to the object such that, given the maximal available distance, the required impact momentum can be achieved. The procedure then attempts to add auxiliary environment object to the mobile object such that its mass is increased by the required amount. This was prototyped in our implementation by iterating through the objects in the environment that the robot can carry and attempting to place them collision-free on the object[9]. If this was successful, the procedure computes the new minimal acceleration distance based on the updated weight and the same verification step as above is called to ensure collision-free motions. If successful, the object to use, distance and list of objects to add to the object prior to acceleration are returned.

5.2.5 Evaluation

We have implemented the proposed algorithm using the dynamic simulator DART [68] and performed 1000 randomized experiments on a $10m \times 10m$ workspace with uniformly sampled number of objects $(3-10)$, object sizes, object weights, object locations as well as required force $(40-80N)$ for jammed inwards opening doors and momentum $(80-120Ns)$ for outwards opening locked doors. For inwards opening doors, the door was set to be jammed at a $30°$ angle. The robot's maximal force was set to $30N$. The environments were

[9]Note that in general it also has to be verified that the objects are rigidly attached such that they do not keep moving upon impact.

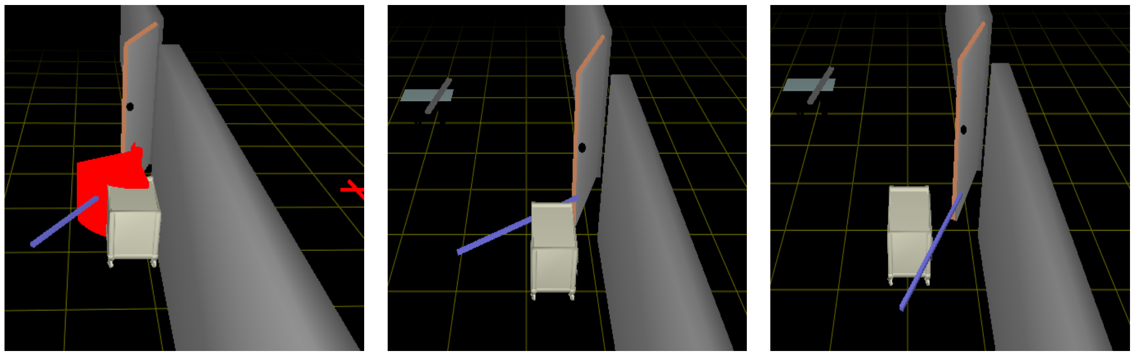

(a) (b) (c)

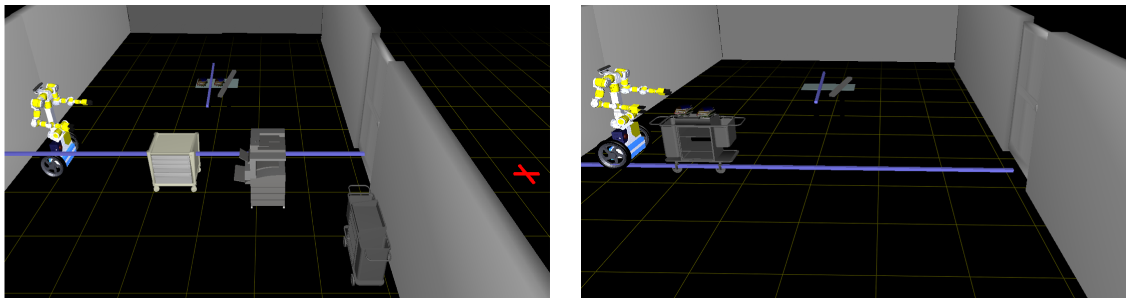
Figure 32: (a) Constraint propagation method with constraint space. (b) Solution for a different run. (c) Rejected configuration based on force direction.

generated such that a solution exists by ensuring that for jammed doors at least one long enough object is presented, and for locked doors either a heavy enough movable object is present or the required weight could be added to a movable object. Fig. 1(a) and 1(b) show example domains.

Depending on the state of the door, the planner attempts to construct an appropriate structure to gain the necessary mechanical advantage: for inwards opening jammed doors the planner attempts to find a lever like structure and for outwards opening locked doors battering ram like structures. We will now iterate on these sub-procedures in turn.

5.2.5.1 Example Solutions

Lever Fig. 32 shows example solutions as well as a rejected sample for finding lever structures for an environment similar to Fig. 1(a). The constraint propagation method determined that in general an edge on the cart could be used as a fulcrum point and attempted to place an edge of the cart in the constraint space. If the algorithm found a solution, it was obtained after an average of 16 samples. Fig. 32(c) shows a configuration that was rejected by the algorithm. The configuration was rejected because the contact force between the cart and the lever was opposing the force direction the robot needed to apply to the lever.



(a) Superimposed direct solutions. (b) Solution obtained by adding books to the cart.

Figure 33: Battering ram example solutions.

Battering Ram Fig. 33(a) shows two separate solutions superimposed for an environment similar to Fig. 1(b). While the small cart needed to be placed further back from the door than the printer, both solutions allowed the robot to achieve the necessary impact momentum.

Fig. 33(b) shows a solution for a similar environment, however without the cart and printer from the previous solution. The only movable object remaining, the grey cart, was not heavy enough to achieve the required momentum within the available space. The determined solution therefore included placing books, originally located on the table, onto the cart. The updated weight of the cart was then used to find the configuration visualized in Fig. 33(b) as a solution. Similar cases occurred in 27% of all solutions requiring battering ram structures for which an average of 50.6kg needed to be added to objects.

5.2.5.2 Failure Cases

While all generated environments were theoretically solvable, only 82% were actually solved by our algorithm. The main failure points occurred during attempts to find a battering ram structure and were caused by objects, other than the one selected, being located in front of the door. This prevented a collision free acceleration of the chosen object.

Additional unsolved environments included failures to find a lever structure before hitting the sample threshold, which was set to 20 in our implementation.

5.2.5.3 Computation Time

If a lever like structure was needed, the algorithm first had to compute the free space region containing the robot in order to construct the constraint space. This was done using a Breath-First-Search with a grid resolution of $20cm$ and took $5.5s$ on average. The remaining construction of the constraint space region then took $1.24s$. An average of 16 configurations of potential fulcrum points and lever objects needed to be tested within the constraint space before a valid solution was found. This took $1.37s$ on average, yielding an overall average computation time of $8.11s$ for finding lever structures. Determining a battering ram structure took an average of $1.4s$.

5.2.5.4 Comparison

To be able to evaluate the usefulness of the constraint propagation method more generally, we implemented the baseline approach described in Section 5.2 for finding lever structures.

Recall that the baseline regression-planning approach involves searching over object configurations sequentially, in the order defined by the constraints. However, even for a single object, the search space is too large for an exhaustive search. We must therefore use heuristics to prioritize the configurations to explore, which are difficult to define given the interdependence between choices of the planner.

The algorithm begins by selecting a specific contact point with the door cp_{dl}. To find a good cp_{dl}, the algorithm samples a set of cp_{dl} candidate points. Fig. 34(a) visualizes an example output of this step. The obtained points are then sorted according to a custom scoring function (the heuristic) defined as a weighted sum of the force required at this point to release the door and the height of the point relative to the robot's workspace. The algorithm then iterates through this sorted list of candidate points and attempts to find a fulcrum point. As a fulcrum point always has to lie on the edge of an object, the algorithm first determines edge points that form a collision-free line with cp_{dl}. To achieve this, an infinitely long lever is placed at cp_{dl} and rotated 360°. Every collision point with an edge during this rotation constitutes a potential fulcrum point. Fig. 34(b) visualizes an example collision. However, fulcrum points can not be on the target object itself and the angle between the lever and the desired force direction has to be roughly 90°. Collision points that do not fulfill these requirements are therefore rejected. Fig. 34(c) shows an example of a collision point that was rejected because of the angle relative to the required force direction.

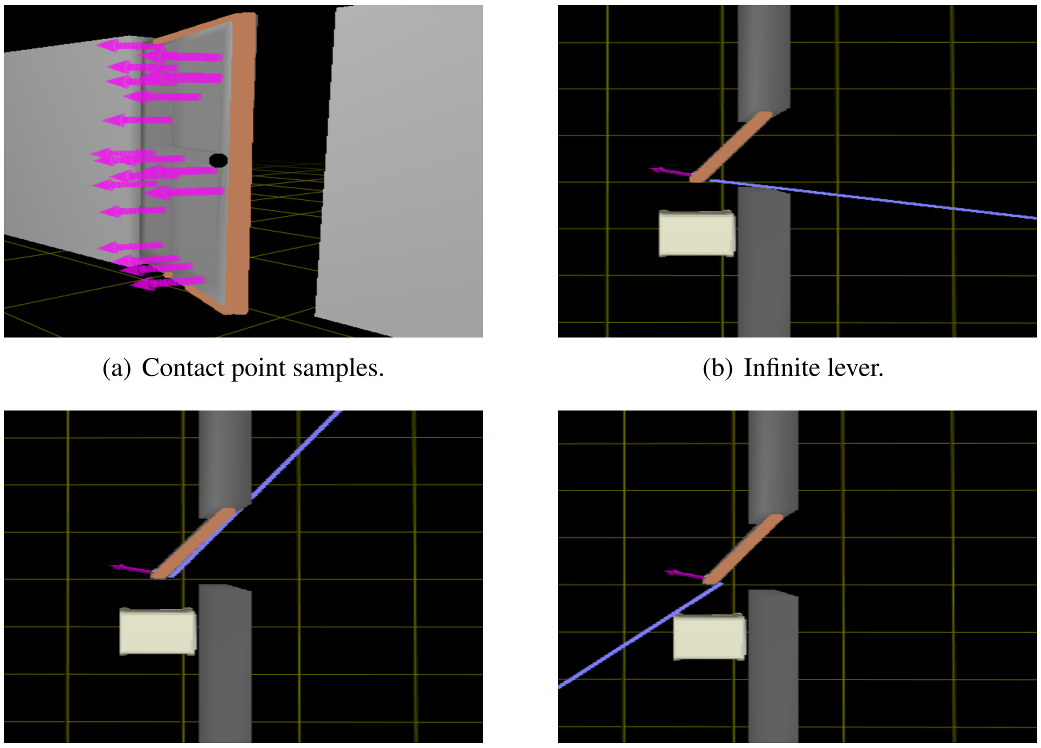
For collision points that are not rejected, Eq. 19 is now used to compute cp_{el}. Eq. 18 then directly yields the minimal length an environment object has to have such that the required mechanical advantage can be achieved. This information is now used to limit the environment objects that are considered to be used as a lever. If any of these candidate object can be fitted, the procedure returns.

This method is conceptually simple and computationally efficient as it is not necessary to construct the reduced search space directly. However, it has two substantial shortcomings:

1. The method does not reason about creating fulcrum points by first moving an object to a different position.
2. It is not obvious where to backtrack to if no solution could be found.

The first shortcoming does not allow the algorithm to solve scenarios as depicted in Fig. 1(b). The second shortcoming makes it difficult to determine that no solution exists as a different choice during the search, such as the contact point with the door, could have permitted a solution. For example, if the initial cp_{dl} was chosen too high in Fig. 34 the cart would not have been considered as a fulcrum point despite yielding a solution. In general, it is not obvious which decision during the planning stage caused a failure and the algorithm potentially has to evaluate multiple choices at each level of the plan.

If a solution not requiring to first move a fulcrum object into position exists, such as in Fig. 34, the algorithm returns a solution within $3.4s$ on average. This is almost 60%


(a) Contact point samples.

(b) Infinite lever.

(c) Rejected sample due to force direction.

(d) Valid fulcrum point.

Figure 34: Example configurations.

faster than the proposed method, which always computes the reduced search space first. However, if no simple solution exists, the methods takes $64.2s$ on average to return as the algorithm backtracks through the entire search tree and iterates through all cp_{dl} samples, 40 in our implementation, before declaring failure.

We can see that our proposed method is outperformed if simple solutions exist, as the computation of the reduced space is computationally expensive. However, our proposed method is able to find solutions for more complex scenarios.

5.2.6 Discussion

In this section we addressed the combinatorial search complexity inherent to multi-object configuration problems by back-propagating physical constraints between useful combinations of objects. Our method is applicable to domains in which the robot has access to all relevant attributes of applicable structures and was successfully applied to the problem of autonomously escaping a room with a jammed door using available objects. The algorithm was able to chose between constructing a lever for prying the door, ramming it with a cart, and even loading the cart with books to attain the required momentum.

While, to our knowledge, this is the first time these reasoning patterns have been exhibited autonomously, much work is still required for a real robotic system to reliably use environment objects as tools. From an algorithmic perspective the system should be able to automatically determine the physical constraints imposed by the task. In addition, the back-propagation of these constraints needs to be generalized. From a practical perspective, the system needs to be able to handle the uncertainty and imperfect knowledge present on a real robotic system. In the following chapter, we therefore present a system that extends the NUMO domain to cases where the robot has substantial initial state uncertainty and only obtains environment information through the use of onboard sensors.

CHAPTER VI

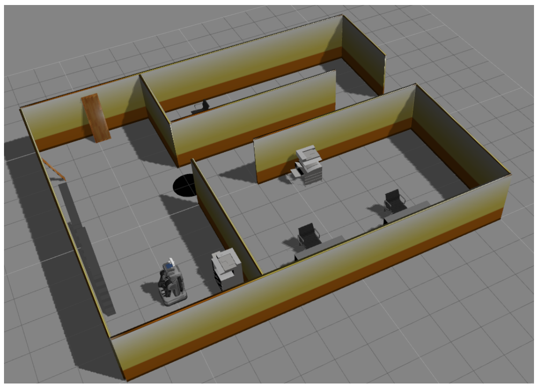
NAVIGATION USING MANIPULABLE OBJECTS WITH ONLY ONBOARD SENSING

The previous chapter introduced the NUMO domain as a generalization of the NAMO domain and presented two complete systems for the cases of overcoming geometric and force constraints. However, both systems assumed full world-knowledge. In this chapter we present a system that allows a robot to reason about using a environment object to overcome an obstruction despite substantial lack of initial state information[1]. The system is able to solve I_1UU class problems (Section 1.2).

Suppose one is trapped in a room with burning gasoline. The human searches the environment for something that can be placed over the gasoline, finds a long enough board, places it over the fire and escapes. Typically, such situations can not be anticipated a-priori, and no predetermined action plan exists. Rather, the present situation and objects have to guide the actions *online*. Possessing such capabilities becomes essential for robots that are expected to operate autonomously in complex, unstructured environments such as disaster areas.

To bring robots closer to such capabilities, this chapter presents the first framework that allows a robot to reason *online* about using an environment object to facilitate its task completion. To understand the value of such a framework, consider the example visualized in Fig. 35(a) in which a robot is tasked with escaping a damaged building. As typical for such scenarios, we assume that the robot has access to a map containing the static environment properties, but has no knowledge about the existence, size or location of non-static environment objects and no knowledge about any potential structural damages. Unaware

[1]This work appears in [62].


(a) Initial environment configuration. The robot is not aware of the location of any non-static environment objects or the hole in the floor, which prevents the robot from escaping the building.

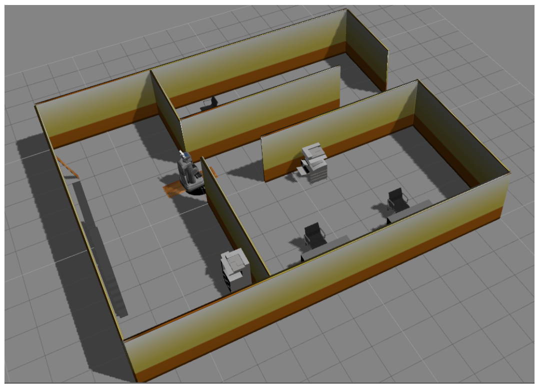

(b) The robot successfully escapes the building by using a board it found in the environment to get over the hole.

Figure 35: Example execution of the proposed framework.

that the floor has been critically damaged, making the only exit unreachable, our system starts by computing a motion plan for the robot to escape the building using the exit. As the robot executes the motion plan it obtains sensor readings about the environment and eventually detects the hole in the floor. Similar to existing systems such as [47], our system re-evaluates the current path based on this new information and realizes that there is no alternative path to exit the building. In contrast to existing work however, our method does not just declare failure and stop the robot. Instead, it guides the robot to evaluate the environment for ways of overcoming the obstruction. For the example in Fig. 35(a), the robot searches its environment, finds a long enough board in the room, places it over the hole and successfully escapes the building. Fig. 35(b) visualizes this successful escape. We are not aware of any other method that would have enabled the robot to escape this situation.

To achieve such behaviors, the proposed framework builds upon execution monitoring (Chapter 4) and constraint relaxed planning (Chapter 5). First, it uses execution monitoring to determine online if the current path is obstructed. If an obstruction is detected, our method utilizes constraint relaxed planning to determine if the obstruction can be avoided or not. For cases in which an obstruction cannot be avoided, the framework determines what properties (*e.g.* dimensions) an object needs to have to be helpful in overcoming the obstruction. The robot is then guided to search the environment for *any* object with these properties. If a suitable object is found, the framework computes the necessary grasp and drop motions and finally guides the robot to use the object to overcome the obstruction.

6.1 Overview

We now provide an overview of the proposed system before discussing implementation details in the following section.

6.1.1 Assumptions

We assume that the world is static and that the robot has access to a map containing immobile environment objects such as walls and stairs. We do not assume that the robot has a-priori knowledge of any structural differences between the real world and the map or of the existence or location of manipulable environment objects prior to any sensing actions.

The framework focuses on cases in which a single object can be used to resolve an obstruction. We also assume that obstructions can be solved independently.

6.1.2 Framework

The framework, as visualized in Fig. 36, is initialized by computing a motion plan for the robot from its current location to the goal configuration. To be able to handle obstructions, the framework uses constraint relaxed path planning (Chapter 5). In contrast to traditional motion planning systems, that either return a sound path or no path at all [52], the constraint relaxed planning system might return a path that violates robot constraints as it treats obstructions as soft constraints rather than hard constraints. Consequently, if no alternative path existed, this planning step could return a path that requires the robot to move over a gap. While such a path would not directly be executable by the robot, constraint relaxed planning provides two crucial insights. First, it establishes whether a low cost path to the goal without obstructions exists. Second, if no such paths exists and obstructions need to be overcome, it provides information to subsequent planning steps about which obstruction

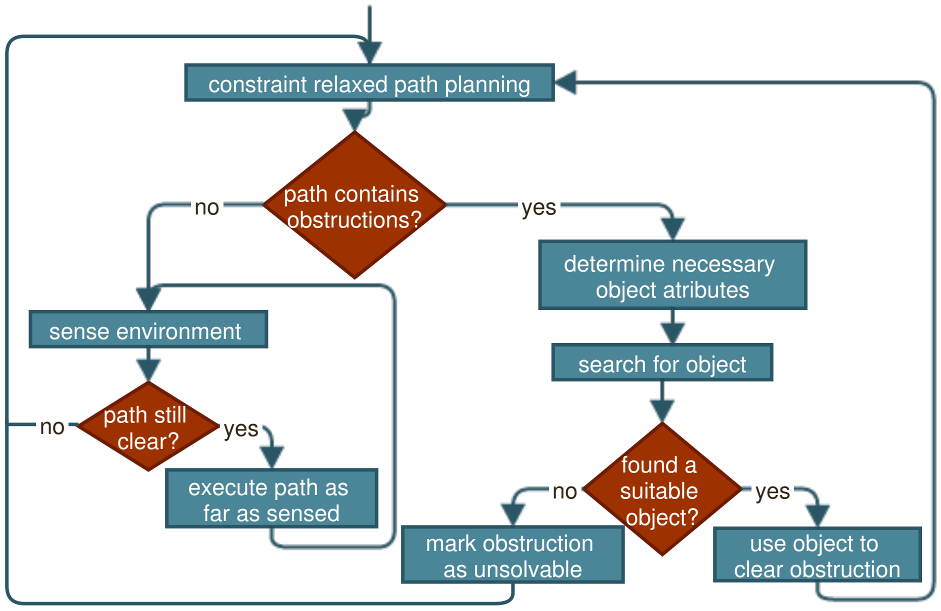
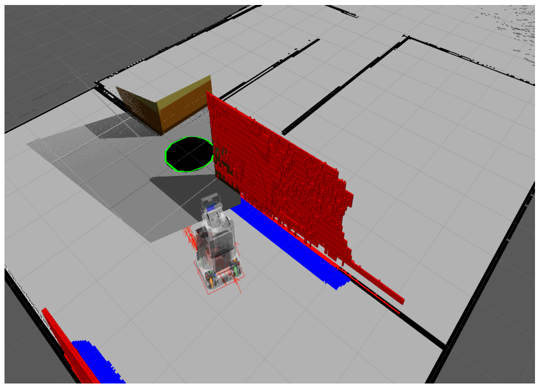

constraint relaxed path planning
path contains obstructions?
no
yes
determine necessary object atributes
sense environment
search for object
no
path still clear?
yes
execute path as far as sensed
found a suitable object?
no
yes
mark obstruction as unsolvable
use object to clear obstruction

Figure 36: Flowchart of the proposed framework.

needs to be cleared. While this property was only utilized to reduce the search space in Chapter 5, the proposed framework explicitly uses this information to gain insight into the kind of object the robot needs to search for in order to be able to clear the obstruction.

To handle the fact that the constraint relaxed planning step could either output a sound path or a path containing constraint violations, the proposed framework branches. If the constraint relaxed planning step finds a path to the goal without obstructions, the robot is tasked with moving along the path while continuously sensing the environment for potential obstructions. In case an obstruction is intersecting the path, the framework guides the robot to evaluate the environment for options to overcome the obstruction.

To find helpful objects in the environment, the framework directly utilizes the output of the constraint relaxed planning step to determine the necessary properties any suitable object needs to have. It then controls the robot to search the environment for such an object. Note that this process stands in contrast to traditional object recognition problems (*e.g.* [69]) in which the task is to find a specific object. Here, the goal is to find *any* object that is usable by the robot. If a suitable object is found, the frameworks proceeds by


(a) Obstruction detection. The green polygon indicates the detected obstruction.

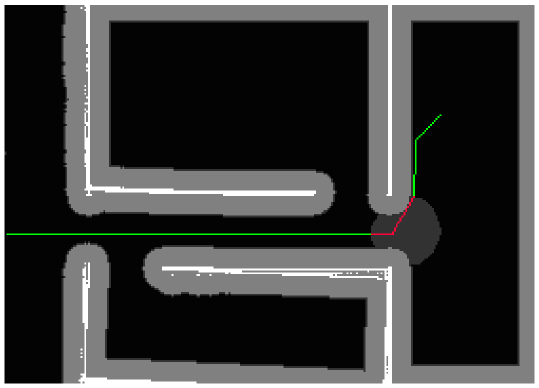

(b) Constraint relaxed planning step output. The red portion of the plan indicates a constraint violation, requiring the robot to resolve the constraint prior to following this part of the plan.

Figure 37: Obstruction detection and constraint relaxed planning step examples.

guiding the robot to use the object to overcome the obstruction. Independent of whether this operation succeeds or fails, the framework loops. This looping mechanism allows the robot to just treat the environment configuration resulting from the repositioning of an object as a new problem instance. In turn, this enables the robot to recover from failure situations in which the usage of an object did not result in the anticipated outcome (*e.g.* a gap is not completely covered).

6.2 Implementation

While the previous section provided a general overview of the proposed framework, we now provide a detailed description of an actual implementation and demonstrate example outputs. Our realization of the general framework described above focuses on cases where ground damages such as holes could prevent the robot from reaching its goal.

The framework requires execution monitoring to ensure that the robot does not collide with previously unknown objects and to detect obstructions. We avoid unknown objects by not allowing the robot to enter any space that it has not previously scanned (e.g. using a Kinect [112]) and for which the scans do not indicate free space. Detecting obstructions, such as structural damages, however, is more challenging. We now detail upon our specific implementation for detecting holes and oil spills.

6.2.1 Detecting Obstructions

To detect crucial ground obstructions, we are utilizing a head mounted Kinect sensor. To support liquid ground obstructions as well as structural damages such as as hole, our implementation utilizes the fact that these kind of obstruction usually result in a very different ground colors than normal ground, and color threshold the Kinects RGB camera data. This allows us to detect obstructions in real-time. In order to not sacrifice 3D information about potential obstructions by restricting ourselves to vision methods, we are associating the color information provided by the Kinect's RGB camera with the depth information obtained by the Kinect's depth sensor. The world coordinates of the obstruction's bounding polygon are then directly given by the 3D information associated with the thresholded pixels.

This process is sufficient for cases where the robot either observes a complete obstruction or no obstruction at all. However, if the robot drives towards an obstruction, each scan could reveal more parts of an obstruction, resulting in a frequently changing bounding polygon and consequently many subsequent calls to the path planning system. To minimize the occurrence of such partial obstruction reports, our implementation is not reporting detected obstructions to the planning system until either the obstruction's dimensions do not increase anymore, or the robot is getting too close to the obstruction. Fig. 37(a) visualizes an example output of this obstruction detection method for the scenario shown in Fig. 35.

6.2.2 Planning

If the obstruction detection algorithm reports an obstruction to the planning system, the obstruction is added to the internal cost map and re-planning is triggered. Similar to Chapter 5, the planner performs an A* search over the discretized representation of the configuration space of the robot but considers collisions with obstructions as soft constraints rather than hard constraints. A heuristic cost penalty is applied for each initial intersection with an obstruction. The A* search returns a motion path for the robot as well as a list of obstructions that have to be resolved to clear the path. Fig. 37(b) shows the output of the constraint relaxed planning step following the detection of the hole in Fig. 37(a). The output indicates that the robot has to cross the hole in order to reach the goal.

6.2.3 Object Search

If the constraint relaxed planning step indicates that an obstruction needs to be resolved for the robot to be able to get to the goal, the framework continues by abandoning the process of attempting to reach the goal through pure navigation. Instead, the robot is now tasked with finding a suitable object in the environment to overcome the obstruction. For example, in the scenario visualized in Fig. 37, the robot needs to find an object to help it cross the hole.

As mentioned above, in contrast to most existing research in object detection, this step does not focus on detecting a known object, but rather on finding *any* object that the robot could utilize to overcome the obstruction. To achieve this, our object detection algorithm is based on the output of the constraint relaxed planning step. Recall that the constraint relaxed planning step returns the exact obstruction that is currently blocking the robot. We can now utilize this information to determine the minimum dimensions for a suitable object and use these dimensions to guide the search. For the example visualized in Fig. 37, we need to find an object that has at least the length of the hole and is at least as wide as the robot base. The robot needs to explore the environment to find such an object. While reasoning about most likely locations of candidate objects is an interesting research area in itself (*e.g.* [108, 109]), our implementation takes advantage of simple heuristics such as on-the spot rotations and wall following to compute exploration waypoints for the robot that cover the entire reachable space as defined by the current cost-map[2].

[2]We do not consider resolving obstructions just to increase the searchable space for the robot.

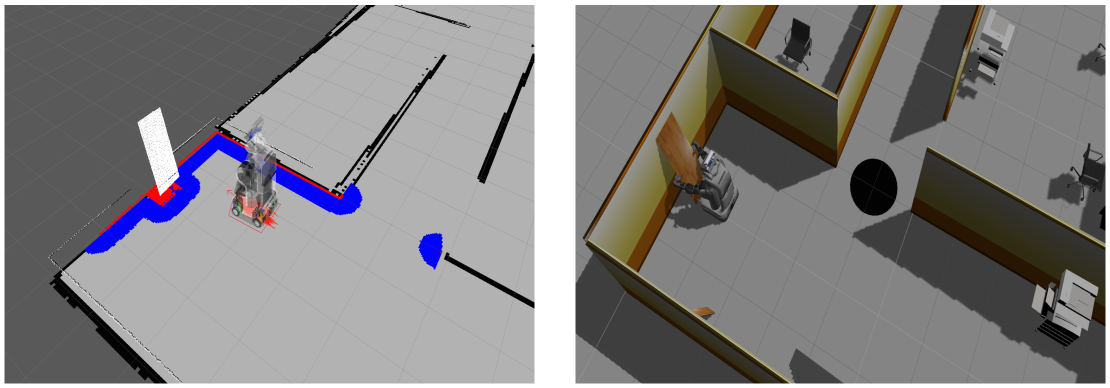

(a) Robot detects suitable obstacle. (b) Robot executes grasping subroutine.

Figure 38: Constraint resolution example.

The robot now proceeds by navigating to the exploration waypoints while scanning the environment for candidate objects using the Kinect's point cloud data. To detect suitable objects, we segment the point cloud data into individual object clusters. This is achieved by using the environment map to remove the ground plane as well as walls from the point cloud prior to clustering. Clusters that do not fulfill the dimensionality requirements, as determined above, are then rejected. Further, any clusters that are above a size threshold, indicating that the robot would likely not be able to manipulate the corresponding object, are rejected. The remaining clusters are then sorted based on a custom cost function. We used a scoring function that attempts to capture the notion of "manipulable" using surface smoothness and object width. In the order defined by the cost function, the robot is now tasked with attempting to use the objects to overcome the current obstruction.

Fig. 38(a) visualizes an example output of this obstacle detection method.

6.2.4 Grasping and Dropping

If a candidate object has been found, the framework computes a navigation plan to the object. When the robot has reached the object grasp position, which in our implementation is determined to be $20cm$ in front of the cluster, our implementation computes a detailed motion plan to grasp the object. We use the cluster information to determine pre-grasp configurations for the grippers. These configurations are computed to be $5cm$ from the edges of the cluster on each side. Given these pre-grasp configurations, the system performs a RRT-connect search with smoothing [50] for each arm individually and moves the grippers into those configurations. The arms are then controlled to move the grippers inwards until contact with the object is established. Upon contact, the grippers are closed and the shoulder joint of each arm controlled to lift the object from the ground.

If the object is successfully grasped, the algorithm computes a motion plan to guide the robot to the obstruction and align the robot with the obstruction. The alignment is determined based on the initial path direction. If the robot has reached this drop location, the robot executes a drop motion. We implemented the drop motion as a reversion of the grasp motion with the addition of a slight motion of the robot base in the direction of the obstruction. This is done to ensure that the object falls in the correct direction. If the drop was successful, the algorithm marks the new location of the object as traversable space and re-starts the constraint relaxed planning system. Fig. 35(b) shows the behavior of the robot after the successful drop of the object.

6.3 Evaluation

We implemented the proposed framework in simulation on the PR2 robot using Gazebo and ROS [78]. We performed multiple runs on environments similar to Fig. 35 with varying start locations of the robot and varying positions of the objects.

Prior to running the experiments, we generated a map of the static environment properties by teleoperating the robot in the empty environment and running a SLAM algorithm [102]. For each experiment run, the robot was given access to the according environment map.

6.3.1 Runtime

Obstruction Detection The obstruction detection subroutine took an average of 74*ms*, allowing the robot to constantly monitor the ground in front of it.

Planning The constraint relaxed planning subroutine took an average of 1.7*s* if no obstruction was blocking the goal and an average of 14.6*s* if all paths to the goal were blocked by obstructions. We can observe that if no direct path to the goal existed, this step takes substantially longer. This is caused by the fact that we used a very high penalty for initial path intersections with obstruction to avoid unnecessary environment modifications. Consequently the planner explored a larger portion of the space before intersecting obstructions. If no known obstructions were disconnecting the robot from the goal, the planner could explore the space more efficiently. This behavior is similar to what was observed in [64].

Object Search Evaluating the point clouds for potential candidate objects to resolve a given obstruction took an average of 2.72*s*. In our experiments, the robot needed to repeat this procedure an average of 43 times at different exploration poses before finding a suitable object.

Object Grasp and Drop Planning for object grasp motions took an average of 1.3*s*. As the drop motions were implemented as a reversion of the grasp motion, no planning was necessary for dropping the objects.

6.3.2 Failure Cases

While all our experimental setups where solvable in principal, we encountered failure cases. First, we encountered task level failures due to insufficient grasps. As our implementation computes grasp configurations solely based on the cluster information and does not reason about force closure or stable grasp configurations, the object occasionally slipped during locomotion. Frequently, such slippage caused the object to fall into a configuration that did not allow the robot to pick the object up again. If no other object was available in the environment, the robot was not able to exit the building, resulting in task level failure.

Second, the drop motion of the object would not always result in a satisfactory positioning of the object. While, as discussed above, the framework can handle such cases to some degree due to the fact that the algorithm loops and the robot just treats the current environment configuration as a new problem instance, we encountered cases in which the object would critically block the hole, again resulting in task-level failure.

We anticipate that future iterations of our implementation will deploy more sophisticated grasp and drop methods.

6.4 Discussion

To our knowledge, this chapter presented the first framework that allows robots to reason *online* about using the environment to make progress towards a goal that is not directly reachable. The framework builds upon execution monitoring and constraint relaxed planning to detect potential obstructions and uses manipulation planning to help the robot overcome critical obstructions. We presented a complete implementation of the framework in realistic simulation on the PR2 robot, allowing it to solve previously unsolvable problems.

The framework reasons from the current obstruction the robot is facing to the kind of object properties needed to overcome the obstruction. The mapping from object properties to possible actions can be regarded as an affordance mapping [28]. In contrast, the introduced framework identifies ”inverse affordances” - mappings from desired actions to required object properties for task completion. We expect future research to extend upon this insight.

CHAPTER VII

CONCLUSION

In this thesis we showed that robots can autonomously modify their environment to achieve task completion in the presence of lack of support for mobility, the need to increase force capabilities and partial knowledge. As an initial step, we demonstrated how the Navigation Among Movable Obstacles (NAMO) domain, which allows a robot to autonomously move objects out of its way, can be extended to cases with incomplete a-priori environment knowledge. We introduced a NAMO planner that, for cases in which adjacent free-space regions can be connected by manipulating a single object, allows a robot to take its current degree of uncertainty about relevant object properties into account when deciding on which objects to move out of its way. Building on these results, we then showed how a robotic system can perceive partially occluded objects using onboard sensing and use such capabilities to perform NAMO even with substantial uncertainty in sensing and action as well as fundamental lack of information about the initial state. The system was successfully executed on a real robot.

While the extension of the NAMO domain to the more realistic setting of state and action uncertainty is a major step towards fully autonomous robots capable of operating in common household environments or even disaster areas, this thesis also pointed out that it may not always be sufficient to just reason about moving objects out of the way. If the robot is separated from the goal through a gap it cannot cross, or by a stuck door that it cannot open directly, no amount of reasoning about moving objects out of the way will allow the robot to reach its goal. We therefore generalized the NAMO domain to the Navigation Using Manipulable Obstacles (NUMO) domain which enables a robot to reason about *using* environment objects as tools.

We presented multiple NUMO realizations for the cases of geometry or force constraint limitations. First, we resented a system enabling a real humanoid robot to autonomously use environment objects to build itself a bridge and a stair step. The system is applicable to situations in which a single object can be used to overcome an obstruction. Second, we introduced a framework allowing a robot to use existing knowledge about lever and battering ram structures to reason about using environment objects to open a stuck or locked door. Finally, we showed how the NUMO domain can be extended to operate *online* using onboard sensing.

We now outline some future research directions before concluding with final remarks.

7.1 Future Work

Extending NAMO-MDP problem classes. The NAMO-MDP as presented in this thesis is currently restricted to cases where free-space regions are disconnected by a single object and free-space connections can be reasoned about independently. This is an instance of the I_1UL problem class as defined in Section 1.2.1.1. It would be interesting to investigate if the NAMO-MDP can be extended to support more complex cases.

Reason about most likely position of objects. Our framework for NUMO with lack of initial state knowledge takes advantage of simple heuristics to determine where to look for environment objects that could be used to overcome an obstruction. To allow for scaling to larger environments, it would be beneficial if the system would reason about most likely locations of object. It should be investigated if systems such as [108, 109] can be generalized to reason about the location of suitable objects.

Discovering object attributes. We presented a NUMO system that allows a robot to reason about using environment objects despite substantial lack of initial state information and relying only onboard sensing. The system selected the first object the robot detected that was large enough to overcome the constraint at hand. However, a robot typically cannot directly perceive material properties of an environment object. It should test an object's applicability to the task before deciding to use it. For example, a robot should test the strength of a board by attempting to bend it before deciding to use it to cross a gap. In more general terms, algorithms for the NUMO domain should support reasoning about information gathering actions. It would be interesting to investigate methods for determining the most suitable information gathering action, such as [25], within the NUMO domain.

Reasoning about multiple object solutions. The NUMO domain realizations presented in this work are mostly restricted to cases where constraints can be resolved through the use of a single object. However, it is not always possible to find a single object in the environment to sufficiently change the environment topology. Instead, the robot should be able to reason about creating entire structures or simple machines from objects found in the environment.

Discovery of required mechanical advantage. Our system for NUMO domains with force requirements should be generalized. The presented system requires substantial pre-programming of physics knowledge for the problem at hand. Ideally, the robot should be able to autonomously determine the necessary mechanical advantage for a given scenario. It would be interesting to investigate the use of a general purpose physics engine to automatically recover the required mechanical structure. Further, to execute the plans determined by the planner on a real robot system, additional forces such as friction at the fulcrum should be considered.

Action uncertainty in NUMO. The NUMO systems presented in this work are able to handle action uncertainty to some extent through re-planning. However, as shown for the NAMO domain, it is beneficial if the robot can reason about the uncertainty of interacting with a specific object. A robot should prefer the use of a rod that it is certain is strong enough over the use of a rod that it believes might break during use. To achieve such a behavior, it would be interesting to investigate the applicability of the NAMO-MDP framework to NUMO cases.

7.2 Final Remarks

The reasoning methods described in this work are vital for robots to operate autonomously in unstructured environments. If in the near future robots are to replace humans in dangerous search and rescue missions, assist humans in household environments, work at construction sides or operate more autonomously over large distances, robots need to be able to get to their goal by all means necessary. A robot that is kept from reaching a victim or critical instruments and tools just because it is blocked in by easily movable objects is not a valuable replacement for a human counterpart. In fact, just like humans, a robot should even be able to reason about ways of using environment objects to create itself a traversable path if none exists. Robots should be able to do Navigation Using Manipulable Objects. We expect that future work based on the concepts presented in this thesis will allow robots to achieve the intelligent tool-orientated behavior which is characteristic of human beings.

APPENDIX A

PLANAR-GRAPH CONSTRAINT

In Section 3.3.4.1 we exploited a standard result regarding planar graphs, which we reproduce here for completeness.

First, recall Euler's formula, which places a constraint on the number of possible edges e, vertices v, and faces f in a planar graph:

$$v - e + f = 2 \tag{26}$$

Now consider the set of all possible *edge-face* pairs $p \in P$, (for $v > 2$) where an edge is added to P for each face in which it appears. Observe that since an edge can contribute to at most 2 faces, we have $|P| \leq 2e$. In addition, each face must have at least 3 edges, implying $|P| \geq 3f$. Plugging this into Eq. 26, we have $e \leq 3v - 6$.

REFERENCES

[1] "Fukushima accident 2011," *World Nuclear Association*, 2011.

[2] "pybox2d." http://code.google.com/p/pybox2d/, June 2012.

[3] ALTEROVITZ, R., GOLDBERG, K., and OKAMURA, A., "Planning for steerable bevel-tip needle insertion through 2d soft tissue with obstacles," in *IEEE International Conference on Robotics and Automation*, 2005.

[4] ATKESON, C. G., MOORE, A. W., and SCHAAL, S., "Locally weighted learning," *Artificial intelligence review*, vol. 11, no. 1-5, pp. 11–73, 1997.

[5] BARTO, A. and MAHADEVAN, S., "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[6] BESL, P. and MCKAY, N., "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, 1992.

[7] BICICI, E. and ST AMANT, R., "Reasoning about the functionality of tools and physical artifacts," tech. rep., Department of Computer Science, North Carolina State University, 2003.

[8] BLODOW, N., MARTON, Z., SOOS, A., and BEETZ, M., "Towards 3D object maps for autonomous household robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3191–3198, Oct. 2007.

[9] BONET, B. and GEFFNER, H., "Planning under partial observability by classical replanning: Theory and experiments," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011.

[10] BRAFMAN, R. I. and SHANI, G., "Replanning in domains with partial information and sensing actions," *Journal of Artificial Intelligence Research*, vol. 45, 2012.

[11] BRATMAN, M., *Intention, Plans, and Practical Reason*. The David Hume series, CSLI Publications, 1987.

[12] BRATMAN, M. E., ISRAEL, D. J., and POLLACK, M. E., "Plans and resource-bounded practical reasoning," *Computational Intelligence*, vol. 4, pp. 349–355, 1988.

[13] BROOKS, R. A. and LOZANO-PEREZ, T., "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Transactions on Systems, Man and Cybernetics*, no. 2, pp. 224–233, 1985.

[14] CHEN, H., SHENG, W., XI, N., SONG, M., and CHEN, Y., "Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing," in *IEEE International Conference on Robotics and Automation*, 2002.

[15] CHEN, P. and HWANG, Y., "Practical Path Planning among Movable Obstacles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 444–449, 1991.

[16] CHESTNUTT, J., NISHIWAKI, K., KUFFNER, J., and KAGAMI, S., "An adaptive action model for legged navigation planning," in *IEEE-RAS International Conference on Humanoid Robots*, 2007.

[17] COOK, G. E., "Robotic arc welding: research in sensory feedback control," *Industrial Electronics, IEEE Transactions on*, no. 3, 1983.

[18] DAVIES, B., HARRIS, S., LIN, W., HIBBERD, R., MIDDLETON, R., and COBB, J., "Active compliance in robotic surgerythe use of force control as a dynamic constraint," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 211, no. 4, 1997.

[19] DEMAINE, E., O'ROURKE, J., and DEMAINE, M. L., "PushPush and Push-1 are NP-hard in 2D," in *In Proceedings of the 12th Canadian Conference on Computational Geometry*, pp. 211–219, 2000.

[20] DIETTERICH, T., "An overview of MAXQ hierarchical reinforcement learning," *Abstraction, Reformulation, and Approximation*, pp. 26–44, 2000.

[21] DIFTLER, M. A., CULBERT, C., AMBROSE, R., PLATT JR, R., and BLUETHMANN, W., "Evolution of the nasa/darpa robonaut control system," in *IEEE International Conference on Robotics and Automation*, 2003.

[22] DOGAR, M. and SRINIVASA, S., "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, pp. 1–20, 2012.

[23] EPPE, M. and DIETRICH, D., "Interleaving planning and plan execution with incomplete knowledge in the event calculus," in *STAIRS*, 2012.

[24] FIKES, R. E., HART, P. E., and NILSSON, N. J., "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, 1972.

[25] FITZPATRICK, P., METTA, G., NATALE, L., RAO, S., and SANDINI, G., "Learning about objects through action-initial steps towards artificial cognition," in *IEEE International Conference on Robotics and Automation*, vol. 3, 2003.

[26] FRITZ, C. and MCILRAITH, S. A., "Generating optimal plans in highly-dynamic domains," in *Conference on Uncertainty in Artificial Intelligence*, 2009.

[27] GARIMORT, J. and HORNUNG, A., "Humanoid navigation with dynamic footstep plans," in *IEEE International Conference on Robotics and Automation*, 2011.

[28] GIBSON, J., “The concept of affordances,” *Perceiving, acting, and knowing*, 1977.

[29] HADFIELD-MENELL, D., KAELBLING, L. P., and LOZANO-PEREZ, T., “Optimization in the now: Dynamic peephole optimization for hierarchical planning,” in *IEEE Conference on Robotics and Automation*, 2013.

[30] HAUSER, K., “The Minimum Constraint Removal Problem with Three Robotics Applications,” in *Workshop on the Algorithmic Foundations of Robotics*, 2012.

[31] HOFFMANN, J., “Ff: The fast-forward planning system,” *AI magazine*, vol. 22, no. 3, p. 57, 2001.

[32] HOLZ, D., SCHNABEL, R., DROESCHEL, D., STÜCKLER, J., and BEHNKE, S., “Towards semantic scene analysis with time-of-flight cameras,” in *RoboCup 2010: Robot Soccer World Cup XIV*, pp. 121–132, Springer, 2011.

[33] HORTY, J. F. and POLLACK, M. E., “Evaluating new options in the context of existing plans,” *Artificial Intelligence*, vol. 127, 2001.

[34] HOWARD, R., *Dynamic probabilistic systems*, vol. 317. John Wiley & Sons New York, 1971.

[35] HSIAO, K., KAELBLING, L., and LOZANO-PEREZ, T., “Grasping POMDPs,” in *in Proc. IEEE International Conference on Robotics and Automation*, pp. 4685–4692, 2007.

[36] H.WU, LEVIHN, M., and STILMAN, M., “Navigation Among Movable Obstacles in Unknown Environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2010.

[37] JOHNSON, A. and HEBERT, M., “Using spin images for efficient object recognition in cluttered 3D scenes,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 5, pp. 433–449, 2002.

[38] KAELBLING, L. P., LITTMAN, M. L., and MOORE, A. W., “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, pp. 237–285, 1996.

[39] KAELBLING, L. P. and LOZANO-PÉREZ, T., “Hierarchical task and motion planning in the now,” in *IEEE International Conference on Robotics and Automation*, 2011.

[40] KAELBLING, L. P. and LOZANO-PÉREZ, T., “Integrated task and motion planning in belief space,” *International Journal of Robotics Research*, vol. 32, 2013.

[41] KAKIUCHI, Y., UEDA, R., KOBAYASHI, K., OKADA, K., and INABA, M., “Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[42] KAVRAKI, L., SVESTKA, P., LATOMBE, J., and OVERMARS, M., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[43] KEARNS, M., MANSOUR, Y., and NG, A., "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes," *Machine Learning*, vol. 49, 2002.

[44] KHLER, W., *The mentality of apes*. Routledge, 2013.

[45] KOCSIS, L. and SZEPESVARI, C., "Bandit based monte-carlo planning," *Machine Learning: ECML*, 2006.

[46] KOENIG, S. and SIMMONS, R., "Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models," in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pp. 91–122, MIT Press, 1998.

[47] KOENIG, S. and LIKHACHEV, M., "d^* lite," in *Proceedings of the national conference on artificial intelligence*, pp. 476–483, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[48] KOENIG, S. and LIKHACHEV, M., "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, 2005.

[49] KOENIG, S., LIKHACHEV, M., LIU, Y., and FURCY, D., "Incremental heuristic search in AI," *AI Magazine*, vol. 25, no. 2, p. 99, 2004.

[50] KUFFNER JR, J. and LAVALLE, S., "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000.

[51] KURNIAWATI, H., DU, Y., HSU, D., and LEE, W., "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[52] LAVALLE, S. M., *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[53] LAVALLE, S. and KUFFNER JR, J., "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[54] LEMIEUX, C., *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer Verlag, 2009.

[55] LEVEN, P. and HUTCHINSON, S., "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, vol. 21, no. 12, 2002.

[56] LEVIHN, M., IGARASHI, T., and STILMAN, M., "Multi-robot multi-object rearrangement in assignment space," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[57] LEVIHN, M., KAELBLING, L., LOZANO-PEREZ, T., and STILMAN, M., "Foresight and reconsideration in hierarchical planning and execution," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[58] LEVIHN, M., NISHIWAKI, K., KAGAMI, S., and STILMAN, M., "Autonomous environment manipulation to assist humanoid locomotion," in *IEEE International Conference on Robotics and Automation*, 2014.

[59] LEVIHN, M., SCHOLZ, J., and STILMAN, M., "Hierarchical decision theoretic planning for navigation among movable obstacles," in *Workshop on the Algorithmic Foundations of Robotics*, 2012.

[60] LEVIHN, M., SCHOLZ, J., and STILMAN, M., "Planning with movable obstacles in continuous environments with uncertain dynamics," in *IEEE International Conference on Robotics and Automation*, 2013.

[61] LEVIHN, M. and STILMAN, M., "Using environment objects as tools: Unconventional door opening," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[62] LEVIHN, M. and CHRISTENSEN, H., "Getting there: Using environment objects to facilitate task completion in unknown environments," in *Humanoids Workshop on Policy Representation*, 2014.

[63] LEVIHN, M., DUTTON, M., TREVOR, A., and STILMAN, M., "Detecting partially occluded objects via segmentation and validation," in *IEEE Workshop on Robot Vision (WoRV)*, 2013.

[64] LEVIHN, M., NISHIWAKI, K., KAGAMI, S., and STILMAN, M., "Autonomous environment manipulation to assist humanoid locomotion," in *IEEE International Conference on Robotics and Automation*, 2014.

[65] LEVIHN, M., STILMAN, M., and CHRISTENSEN, H., "Locally optimal navigation among movable obstacles in unknown environments," in *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[66] LIHACHEV, M., FERGUSON, D., GORDON, G., STENTZ, A., and THRUN, S., "Anytime search in dynamic graphs," *Artificial Inteliigence*, vol. 172, no. 14, 2008.

[67] LIKHACHEV, M. and STENTZ, A., "Probabilistic planning with clear preferences on missing information," *Artificial Intelligence*, vol. 173, no. 5, pp. 696–721, 2009.

[68] LIU, C., "Dart: Dynamic animation and robotics toolkit." https://github.com/dartsim/dart/wiki, October 2013.

[69] LOWE, D. G., "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[70] LOZANO-PEREZ, T. and WESLEY, M., "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[71] MARTON, Z.-C., PANGERCIC, D., RUSU, R. B., HOLZBACH, A., and BEETZ, M., "Hierarchical object geometric categorization and appearance classification for mobile manipulation," in *IEEE-RAS International Conference on Humanoid Robots*, December 2010.

[72] MOZOS, O., MARTON, Z.-C., and BEETZ, M., "Furniture models learned from the www," *Robotics Automation Magazine, IEEE*, vol. 18, pp. 22 –32, june 2011.

[73] NISHIWAKI, K., CHESTNUTT, J., and KAGAMI, S., "Autonomous navigation of a humanoid robot on unknown rough terrain," in *Intl Symposium on Robotics Research*, 2011.

[74] PARR, R. and RUSSELL, S., "Reinforcement learning with hierarchies of machines," *Advances in neural information processing systems*, pp. 1043–1049, 1998.

[75] PINEAU, J., GORDON, G., THRUN, S., and OTHERS, "Point-based value iteration: An anytime algorithm for pomdps," in *International Joint Conference on Artificial Intelligence*, vol. 3, pp. 1025–1032, 2003.

[76] PLATT, R., TEDRAKE, R., KAELBLING, L., and LOZANO-PEREZ, T., "Belief space planning assuming maximum likelihood observations," in *Robotics: Science and Systems*, 2010.

[77] POLLACK, M. E. and RINGUETTE, M., "Introducing the tileworld: Experimentally evaluating agent architectures," in *Association for the Advancement of Artificial Intelligence*, pp. 183–189, 1990.

[78] QUIGLEY, M., CONLEY, K., GERKEY, B. P., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., and NG, A. Y., "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[79] RASMUSSEN, C. E., "Gaussian processes for machine learning," 2006.

[80] ROY, N., GORDON, G. J., and THRUN, S., "Finding approximate pomdp solutions through belief compression," *J. Artif. Intell. Res.(JAIR)*, vol. 23, pp. 1–40, 2005.

[81] RUSSELL, S. and NORVIG, P., *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

[82] RUSU, R. B., BRADSKI, G., THIBAUX, R., and HSU, J., "Fast 3d recognition and pose using the viewpoint feature histogram," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2155–2162, IEEE, 2010.

[83] RUSU, R. B., MARTON, Z. C., BLODOW, N., HOLZBACH, A., and BEETZ, M., "Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[84] SCHOLZ, J., LEVIHN, M., ISBELL, C., and WINGATE, D., "A physics-based model prior for object-oriented mdps," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1089–1097, 2014.

[85] SHIMIZU, M. and TAKAHASHI, T., "Training platform for rescue robot operation and pair operations of multi-robots," *Advanced Robotics*, vol. 27, no. 5, pp. 385–391, 2013.

[86] SHUMAKER, R. W., WALKUP, K. R., and BECK, B. B., *Animal tool behavior: the use and manufacture of tools by animals*. JHU Press, 2011.

[87] SINAPOV, J. and STOYTCHEV, A., "Toward autonomous learning of an ontology of tool affordances by a robot.," in *AAAI*, 2008.

[88] SOBEK, R. P. and CHATILA, R. G., "Integrated planning and execution control for an autonomous mobile robot," *Artificial Intelligence in Engineering*, vol. 3, no. 2, 1988.

[89] STEDER, B., GRISETTI, G., LOOCK, M. V., and W, "Robust on-line model-based object detection from range images," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4739–4744, Oct. 2009.

[90] STEDER, B., RUSU, R., KONOLIGE, K., and BURGARD, W., "Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries," *IEEE International Conference on Robotics and Automation*, 2011.

[91] STENTZ, A., "The d^* algorithm for real-time planning of optimal traverses.," tech. rep., DTIC Document, 1994.

[92] STILMAN, M. and KUFFNER, J., "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Journal of Humanoid Robotics*, pp. 322–341, 2004.

[93] STILMAN, M., NISHIWAKI, K., KAGAMI, S., and KUFFNER, J., "Planning and Executing Navigation Among Movable Obstacles," in *IEEE/RSJ International Conference On Intelligent Robots and Systems*, pp. 820 – 826, October 2006.

[94] STILMAN, M. and KUFFNER, J., "Navigation among movable obstacles: Real-time reasoning in complex environments," in *IEEE/RAS International Conference on Humanoid Robotics*, November 2004.

[95] STILMAN, M. and KUFFNER, J., "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, 2008.

[96] STILMAN, M., MICHEL, P., CHESTNUTT, J., NISHIWAKI, K., KAGAMI, S., and KUFFNER, J., "Augmented reality for robot development and experimentation," Tech. Rep. CMU-RI-TR-05-55, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[97] STILMAN, M., NISHIWAKI, K., KAGAMI, S., and KUFFNER, J., "Planning and executing navigation among movable obstacles," *Springer Journal of Advanced Robotics*, vol. 21, no. 14, 2007.

[98] STOYTCHEV, A., "Behavior-grounded representation of tool affordances," in *IEEE International Conference on Robotics and Automation*, pp. 3060–3065, 2005.

[99] SUTTON, R., PRECUP, D., and SINGH, S., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[100] T. NISHIZEKI, N. C., *Planar graphs: theory and algorithms*. Elsevier Science Ltd, 1988.

[101] THRUN, S., BURGARD, W., and FOX, D., *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[102] THRUN, S., BURGARD, W., and FOX, D., *Probabilistic robotics*. MIT press, 2005.

[103] URMSON, C., ANHALT, J., BAGNELL, D., BAKER, C., BITTNER, R., CLARK, M., DOLAN, J., DUGGINS, D., GALATALI, T., GEYER, C., and OTHERS, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[104] VAN DEN BERG, J., STILMAN, M., KUFFNER, J., LIN, M., and MANOCHA, D., "Path planning among movable obstacles: a probabilistically complete approach," *Workshop on Algorithmic Foundation of Robotics*, 2008.

[105] VISALBERGHI, E. and TRINCA, L., "Tool use in capuchin monkeys: distinguishing between performing and understanding," *Primates*, vol. 30, no. 4, 1989.

[106] WALSH, T., GOSCHIN, S., and LITTMAN, M., "Integrating sample-based planning and model-based reinforcement learning," in *Proceedings of AAAI*, no. 1, 2010.

[107] WILFONG, G., "Motion planning in the presence of movable obstacles," in *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, (New York, NY, USA), pp. 279–288, ACM, 1988.

[108] WONG, L. L., KAELBLING, L. P., and LOZANO-PÉREZ, T., "Manipulation-based active search for occluded objects," in *Robotics and Automation , 2013 IEEE International Conference on*, pp. 2814–2819, IEEE, 2013.

[109] WONG, L. L., KAELBLING, L. P., and LOZANO-PÉREZ, T., "Not seeing is also believing: Combining object and metric spatial information," IEEE International Conference on Robotics and Automation, 2014.

[110] YOON, S. W., FERN, A., and GIVAN, R., “FF-Replan: A baseline for probabilistic planning,” in *ICAPS*, 2007.

[111] ZELINSKY, A., “A mobile robot exploration algorithm,” *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, 1992.

[112] ZHANG, Z., “Microsoft kinect sensor and its effect,” *MultiMedia, IEEE*, vol. 19, no. 2, pp. 4–10, 2012.