PHYSICS-BASED REINFORCEMENT LEARNING FOR AUTONOMOUS MANIPULATION

A Thesis
Presented to
The Academic Faculty

by

Jonathan K. Scholz

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2015

PHYSICS-BASED REINFORCEMENT LEARNING FOR AUTONOMOUS MANIPULATION

Approved by:

Professor Charles L. Isbell, Advisor
School of Interactive Computing
Georgia Institute of Technology

Professor Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Henrik Christensen
School of Interactive Computing
Georgia Institute of Technology

Professor Michael Littman
School of Computer Science
Brown University

Professor Andrea Thomaz
School of Interactive Computing
Georgia Institute of Technology

Date Approved: 21 August 2015

ACKNOWLEDGEMENTS

This dissertation is the result of a collaboration with three laboratories across two universities. I owe my deepest gratitude to my advisor, Charles Isbell, who in addition to supporting this research directly at Georgia Tech also sponsored a six month rotation at MIT. Charles' faith in my abilities has driven me to excel, in the way that only a brilliant and respected mentor can. At MIT I worked closely with David Wingate on the development of the ideas that would later form the core of my thesis. David's competence in topics across Machine Learning and Cognitive Science was both humbling and inspiring. I am also deeply indebted to Mike Stilman, my first advisor at Georgia Tech, who imparted a rebellious enthusiasm for robotics research that will stay with me for a lifetime.

I would like to thank the members of my thesis committee, Henrik Christensen, Andrea Thomaz, Magnus Egerstedt, and Michael Littman. Their thoughtful feedback and personal excellence has raised my standards for this thesis. Henrik has been a font of practical advice and guidance in finalizing this research and my future career plans. After the tragic passing of Mike Stilman, Andrea immediately offered to sit on my committee, and has always been inviting and thoughtful in her feedback. My meetings with Magnus have been some of the highest information-density discussions of my graduate career. Input from Magnus has forced me to consider my research agenda from a non-machine learning perspective, and I hope I have succeeded in crafting a thesis that appeals to a broader range of disciplines as a result. As my remote committee member, Michael has probably been more influential on my career than he realizes. His reputation precedes him, and since long before we met Michael has provided some of my favorite examples of great research papers, talks, and jokes.

Looking back, I owe a great debt to Rebecca Saxe, who hired me as a research assistant

in social neuroscience right out of college. Simply put, Rebecca was the most passionate and intelligent scientist I had ever met. I have never seen anyone get more excited by new data, or have more temerity in attacking philosophical questions as a scientist. From her I learned that human intelligence had been upgraded from a mystery to a problem, which set me on my path to where I am today.

I am also fortunate to have been surrounded by amazing colleagues at Georgia Tech. Martin Levihn has been my closest collaborator and friend over the course of grad school. Martin has consistently provided invaluable professional and personal feedback, and like the algorithms we developed together, may be rigid sometimes but comes through in times of uncertainty. From Atlanta to Boston to Hamburg, we have somehow managed to negotiate much of the grad school experience together. Like a true scientist, Baris "The Wise" Akgun always seems to have a ready answer to questions across an incredible range of topics, except where to find his keys. Michael "Misha" Novitzky has been in the PhD program with me from beginning to end, and while we make fun of him for his unsolicited hugs, deep down we all appreciate his kindness and generosity. Alexander "Sasha" Lambert has made coming to the lab more entertaining ever since he joined the program last year, and there are few people I trust more to cheer me up after a rough day. Kaushik "Rogers" Subramanian holds the record for the most declined event invitations, but also the most friendly and fruitful discussions about Reinforcement Learning.

I would like to extend a special thanks to the Krang development team, notably Can Erdogan, Ana Huaman, and Nehchal Jindal. Can Erdogan, who led the Krang development team, was always gracious about sacrificing his personal time to help me fix the robot when it suspiciously broke during my usage hours. Ana Huaman graciously allowed Krang to cannibalize Alita when I broke an arm testing compliant manipulation. Nehchal worked closesly with me during the final implementation push for the results in my last chapter, and was far more dedicated and insightful than I could have hoped for.

Many others have contributed to making my time in Atlanta special, including Manohar

Paluri, Karthik Raveendran, Ashley Edwards, Himanshu Sahni, Heni Ben-Amor, Tobias Kunz, Rowland O'Flaherty, Maya Cakmak, and Jonathan Gladin. I also want to thank the members of the Pfunk and Humanoids labs at Georgia Tech, including Luis-Carlos Cobo, Liam MacDermed, Peng Zang, Arya Irani, Nick Depalma, Pushkar Kolhe, Jesse Rosalia, Neil Dantam, Venkata Subramanian, Michael Grey, Saul Reynolds-Haertle, and Eric Huang.

Finally, I would like to thank my parents and my girlfriend Amber Numamoto. Amber has been loving, selfless, and patient as I worked to finish my PhD, and our discussions ranging from "Beige-ian" statistics to mysticism have helped me see my work and myself from a broader perspective. My father, Bill, shares my enthusiasm for technology, and always makes me feel that my research is fascinating and important. Last but not least, I would like to thank my mother Beatrice for her unwaivering support and guidance for the past 32 years. She has always challenged me to be unconventional, and believed I could do anything from building a car to sustaining myself for four months in Italy. As my first mentor, role model, and advocate, she gave me the strength and self-reliance without which I could not have finished this PhD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

With recent research advances, the dream of bringing domestic robots into our everyday lives has become more plausible than ever. Domestic robotics has grown dramatically in the past decade, with applications ranging from house cleaning to food service to health care. To date, the majority of the planning and control machinery for these systems are carefully designed by human engineers. A large portion of this effort goes into selecting the appropriate models and control techniques for each application, and these skills take years to master. Relieving the burden on human experts is therefore a central challenge for bringing robot technology to the masses.

This work addresses this challenge by introducing a *physics engine* as a model space for an autonomous robot, and defining procedures for enabling robots to decide when and how to learn these models. We also present an appropriate space of motor controllers for these models, and introduce ways to intelligently select when to use each controller based on the estimated model parameters. We integrate these components into a framework called Physics-Based Reinforcement Learning, which features a stochastic physics engine as the core model structure. Together these methods enable a robot to adapt to unfamiliar environments without human intervention.

The central focus of this thesis is on fast online model learning for objects with *under-specified* dynamics. We develop our approach across a diverse range of domestic tasks, starting with a simple table-top manipulation task, followed by a mobile manipulation task involving a single utility cart, and finally an open-ended navigation task with multiple obstacles impeding robot progress. We also present simulation results illustrating the efficiency of our method compared to existing approaches in the learning literature.

CHAPTER I

INTRODUCTION

> It is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects. [67]
>
> Leslie Kaelbling (MIT)

This work focuses on robot manipulation in unfamiliar environments, such as homes, offices, and public spaces. In designing planning and control systems for these environments, a common theme emerges: The modern world was designed for people, not robots. Even simple scenes like an office present vast complexity, and unlike controlled factory or laboratory settings, no two rooms are exactly alike. However, humans are adept at quickly understanding the physical behavior of natural scenes by drawing on extensive prior knowledge [10]. The goal of this research is to equip robots with similar prior knowledge and inference methods to enable them to learn about unfamiliar environments from interaction.

This research agenda raises numerous questions about the form of this prior information, and what principles it encodes. As many researchers in Artificial Intelligence [43, 67] and Cognitive Science [24, 154] have suggested, *objects* are a fundamental concept for physical agents. In principle, we would like our model to capture as broad a space of object behaviors as possible, while simultaneously being data-efficient and tractable to estimate. Unfortunately the space of possible object behaviors is vast, and could draw on techniques from rigid body motion to fluid mechanics, elastics, and even thermodynamics, just to name a few. This research focuses on rigid body dynamics as a starting point for constructing *learnable* models of natural object dynamics. Rigid-body physics is a well-studied field, and the core principles have already been encoded in a usable computational

tool: the *physics engine*. We will argue that these engines provide a compact encapsulation of a range of physical phenomena, and therefore provide an attractive model space for a Reinforcement Learning agent, such as a robot, which has to interact with the physical world.

While intuitive, the underlying principles behind this approach are at the center of debates in both robotics and cognitive science. Within the robotics community, proponents of non-parametric learning methods argue that parametric approaches are too inaccurate and brittle [118]. Within the cognitive science literature, detractors argue that physical simulation fails to explain human success and biases in physical reasoning tasks [31]. While evidence supports these claims in many cases, we argue that physics engines offer good approximations of a large array of useful phenomena, and are compact, conceptually simple to work with, and can be learned efficiently. We will demonstrate these properties by defining a Reinforcement Learning framework utilizing a subset of the modeling API of a 2D physics engine. The approach we take in this thesis constitutes a proof-of-concept using a planar manipulation task and restricted subset of a 2D physics engine. Chapter 7 will consider the problem of extending this approach to a broader range of possible environments and dynamics.

1.1 Motivation

As evinced by recent successes such as the Darpa Robotics Challenge (DRC, [127]), robots are now capable of quite sophisticated behaviors, including autonomous mapping and navigation, manipulation of large objects in clutter, and avoiding dynamic obstacles [106]. These abilities are critical in the vast majority of human environments which, in contrast to factory settings, are only loosely structured and can change over time in ways that are potentially difficult to predict.

These sorts of *semi-structured* environments are also ubiquitous: According to US Energy Information Administration reports, the amount of residential square footage of buildings in the United States outnumbers industrial square footage by a factor of more than 100 (256B vs. 1.9B) [1, 2]. Further, the *modeling* of semi-structured environments is a large part of the effort that goes into deploying robotic systems. A recent robotics-education study found that 56% of the time spent developing a working system focused on behavior and structure modeling, and only 24% on robot construction and 20% on actual software implementation [132].

This issue becomes even more pronounced as we progress to higher-level tasks with humanoid robots. Many cutting-edge research applications on humanoids depend on rich simulation environments for decision making. For example, the core planning algorithm in [96] utilizes a rigid body physics engine to reason about mechanical advantage for different manipulation actions. These methods require carefully designed geometric and dynamic models of the scene and the robot itself, and explicitly utilize these parameters for planning.

In other words, from an autonomous-agent perspective these methods implicitly assume a physics simulator as a model representation already. It is perhaps surprising then that much of the model learning literature in robotics focuses on more general methods such as Gaussian Processes [35] that do not feature *objects* as a core concept. This may be justified by their focus on manipulator and single-object systems. However, it is difficult to imagine generalizing these methods to mobile manipulation tasks that make explicit use of physical concepts such as collisions or mechanical advantage[1]. By contrast, our approach is based on the simple idea of using the same models for learning that applications researchers already use for planning in these domains. We aim to strike a balance between the compactness of differential-equation based models and the generality of more

[1]The closest possible exception with Gaussian Process models is PILCO, which was shown to handle collisions and multiple objects in [36]. However, this required specific choice of shaping potentials for pushing the manipulator away from obstacles, and did not expose parameters for physical reasoning in the manner of a physics engine.

abstract models in Machine Learning. In this thesis we will argue that this approach is more intuitive, requires less supervision, and as we will show, more efficient.

1.2 Approach

This thesis focuses on autonomous model learning from exploration. Our primary interest is mobile manipulation planning, which requires modeling scenes containing multiple interacting objects. Compared with perceiving or planning, interacting with objects with a robot is expensive: it takes considerable time and energy, and invariably risks damaging the robot, the object, or even injuring nearby people [34]. Therefore the particular challenge we consider is how to build models that make efficient use of the data available to the robot.

This challenge can be reframed as a problem of inductive bias: How can we build models that generalize appropriately from limited amounts of data? The methods we present draw from both the robotics and machine learning literatures. We use Reinforcement Learning (RL) as a framework to formalize the learning and planning problem, which offers several benefits for this research:

- It gives a rich theoretical foundation to the decision-making problem under uncertainty in perception, actuation, and even the robot's internal model of the world.
- It offers a clean separation of what the engineer provides, in the form of models and algorithms, from what the end-user provides, which is simply performance feedback.
- It treats the robot not just as a dynamical system, but as a *persistent agent* in the world, complete with internal state and beliefs.

RL is thus an ideal choice for the long-horizon manipulation tasks we consider here, in which information about the world is incrementally revealed to the robot.

Our main contribution is a model-space for a Reinforcement Learning agent that is built on a full rigid-body simulation engine. The benefits of our physics-based approach are both statistical and practical: First, it encodes much of what has been learned about the

mathematical structure of object dynamics, and yields a compact parametric description of non-linear, discontinuous behavior that is easy to learn from small amounts of data. Second, it gathers all model information into a single coherent structure, the physics engine prior. In addition to being parsimonious, this makes it straightforward to implement model learning, perception, and planning algorithms against a single API.

We approach this problem in stages. First, we will introduce a simplified RL framework for model-based planning for multiple objects. This work will serve as an introduction to multi-object planning with empirical object models. Next we generalize the planning methodology to handle a full-size humanoid robot, and incorporate kinematic constraints between the robot and the target object. This work will introduce the basic control methodology for mobile manipulation with *unconstrained* objects, and show how the articulation capabilities of the arms can be used in a long-horizon planner. We then consider the model learning problem, consider the problem of how to learn the dynamics of *constrained* objects. It is in this chapter that we introduce Physics-Based Reinforcement Learning (*PBRL*), and demonstrate its viability and efficiency on real and simulated data.

Finally, we show how these methods can be combined to solve challenging mobile manipulation tasks in unfamiliar environments. We present a full scale manipulation problem called Navigation Among Movable Obstacles (NAMO) which requires carefully sequencing multiple manipulation actions on (possibly) constrained objects to clear a collision-free path to a desired goal position. This task requires a full dynamic model of all objects in a scene, actions for articulating objects with respect to model constraints, and methods for detecting and resolving inaccuracies in these models.

1.3 Thesis Overview

We now provide a concise statement of our thesis and contributions.

1.3.1 Thesis Statement

Physics-based Reinforcement Learning is a feasible and efficient method for autonomous robot manipulation, and enables adaptive behavior in natural environments.

1.3.2 Summary of Contributions

The contributions made by this thesis can be summarized as follows:

- *TS-RRT* — Task-Space RRT is an algorithm for multi-object manipulation planning that operates entirely in the space of object poses (the robot is not represented). *TS-RRT* extends the Rapidly-Exploring Random Tree (RRT) algorithm, a Monte-Carlo planner designed for high-dimensional problems such path planning for a robot manipulator [91] However, instead of requiring a specific goal state like RRT, *TS-RRT* can operate with a reward signal like a Reinforcement Learning algorithm.

- *Articulated Mobile Manipulation* — introduces the ability to articulate a grasped object about arbitrary points in the reachable workspace, and a planner that can operate over the full robot-object system, represented as a deformable footprint.

- *OO-LWR* — an object-oriented non-parametric regression model for capturing multi-body dynamics. *OO-LWR* avoids strong parametric assumptions present in the physics-engine based approach, but makes less efficient use of data.

- *PBRL* — Physics-Based Reinforcement Learning is a model-based RL framework specifically designed for physical planning tasks such as robot manipulation.

1.3.3 Document Outline

The remainder of this document is structured as follows:

Chapter 2 - Background provides background material on the core methodology and related work in Reinforcement Learning and Robotics.

Chapter 3 - Planning with Data-Driven Models introduces the online model-based planning paradigm, and the TS-RRT.

Chapter 4 - Planning with Kinematic Constraints discusses the challenge of bimanual manipulation planning with a full humanoid robot.

Chapter 5 - Learning Kinematic and Dynamic Constraints introduces *PBRL*, which provides a way to learn the dynamics of certain types of constrained bodies from data gathered during manipulation.

Chapter 6 - Learning and Planning with Dynamic Constraints generalizes the articulation abilities of Chapter 4 and incorporates the physics-based model learning framework from Chapter 5 to solve a NAMO problem in an office environment with multiple unknown objects.

Chapter 8 - Conclusion offers closing remarks and perspective on future research directions.

CHAPTER II

BACKGROUND

In this chapter we provide an overview of the core mathematical ideas in this thesis, and summarize the relevant results from the machine learning and robotics literature. Our main interest is in controlling large physical robots in unfamiliar environments, so we focus on methods that relate to learning and planning with data driven models. This section provides a general overview of the main research issues, and we will introduce additional background as necessary in each chapter.

2.1 Physical Scene Understanding

Before delving into the formal methods involved in this thesis it is worth drawing attention to the broader scientific discussion of physical scene understanding. From a sensory perspective, the sheer volume of information available to a physically-grounded agent is daunting. However, from a young age, humans are capable of remarkably nuanced predictions, *e.g.* that a broom propped against a door will fall if the door is opened, or that two colliding objects will reverse directions and impart energy in proportion to their *mass* [137].

Many models have been proposed in both cognitive science to explain these abilities Sanborn et al. argue that a probabilistic physics program, or "Noisy Newton", is cleaner than the heurisitc alternatives, and matches empirical data. Battaglia et al. go further, arguing that while promising, *Noisy Newton* only addresses "idealized cases, much closer to the examples of introductory physics classes than to the physical contexts people face in the real world" [10]. Missing, they argue, is a single parsimonious explanation of how agents can efficiently make sense of static and dynamic object behavior. Instead they suggest that these reasoning abilities can best be explained by an "intuitive physics engine" (IPS), which is similar to a regular physics engine but with probabilistic beliefs on the engine

parameters. In [10], the authors show that subjects' successes and failures at a collection of static and dynamical predictions involving toppling Jenga towers closely agree with the posterior samples from the IPS[1].

Battaglia et al therefore make a strong commitment to the "engine hypothesis", arguing that despite its superficial complexity, it is in fact simpler than heuristic or "feature-based" alternatives when considering the full range of predictions that it must support. In this thesis we adopt a similar view: a physics engine *does* constitute a large amount of domain knowledge to build into an agent, but in the context of object manipulation in general it is simpler than the leading alternatives. Where [10] resorted to human testing to support this view, we will show it quantitatively in terms of agent performance on several tasks.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a canonical formalism for describing decision making problems for dynamical systems [68, 165]. RL is best thought of as a class of problems which can broadly be explained as follows. An "agent", the actor in the dynamical system, interacts with a world represented by a (possibly infinite) set of states; For any state in which the agent finds itself, it selects an available action, observes a new state, and obtains some reward. It is a learning problem because the machinery the agent uses to select actions is typically under-specified, and the agent must adapt from experience. RL is therefore a trial and error approach that leaves room for any sort of structure we may wish to impose to help the agent maximize its rewards.

The two central structures of interest in most Reinforcement Learning algorithms are the *value function* and the *policy*. These are complementary objects – the value function $V(s)$ describes the *expected utility* for any state s in the world, and the policy $\pi(s)$ indicates which action to select in those states. The agent's goal is generally to find the optimal

[1]Posterior samples from an IPS were obtained in the same MCMC fashion as in our approach for *PBRL* in Section 5.3.2

policy π^* that maximizes the long-term expected rewards V^* for every state. Note that $V^*(s)$ is a proxy measure of the agent's future experience – it indicates the total expected reward that can be obtained in state s assuming the agent follows π^*.

Given an optimal policy or value function the agent has everything it needs to behave optimally in the world. Much of RL research has focused on helping the agent efficiently learn the value function, *e.g.* with appropriate function approximators [19, 20], or the policy, *e.g.* with policy gradient methods [166]. However, these functions are often difficult to compute in practice, particularly on problems with large state and action spaces as we consider in this thesis. An alternative approach is to focus the agent's effort at learning the world *dynamics*, which it can then use to select actions by internally simulating their effects. In this thesis we largely focus on providing the agent with an appropriate model representation, such that it can quickly figure out the domain dynamics and plan for itself. However, in Chapter 6 we do introduce a hierarchical structure for approximating the value function of a large MDP.

2.2.1 Markov Decision Processes

The Markov Decision Processes is a mathematical formalism on which much of the Reinforcement Learning literature is based. An MDP is defined by $\langle S, A, T, R, \gamma \rangle$ for a state space S, action space A, transition function $T(s,a) \rightarrow P(s)$, reward function $R(s) \rightarrow \mathscr{R}$, and discount factor $\gamma \in [0,1)$. The agent tries to find a policy $\pi(s) \rightarrow a$ which maximizes long-term expected reward $V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$ for all s.

In model-based RL, the agent is uncertain about some of the components of the underlying MDP and must refine its knowledge from experience. If information about parameters is represented in parametric form and updated in accordance to Bayes' rule with new information, it is referred to as Bayesian RL (BRL) [178]. In this thesis we are primarily interested in the transition model T, and will consider several possible model priors $P(T)$. The overall object manipulation problem then is to use observed transition samples

to update model parameters, and plan using a learned T.

2.2.2 Dynamics Models in RL

Researchers have been exploring different ways of capturing structure in MDP models for decades. In this section we provide an overview of several relevant approaches to modeling the world dynamics for an RL agent. Each approach makes different assumptions about the nature of the model, and exploits this structure to accelerate model learning in different contexts. Our presentation is roughly sorted along a spectrum of decreasing generality and increasing efficiency. We note however that "efficiency" is context-dependent, and we conclude with two models which are efficient in different ways – object-oriented models for *relational* dynamics and regression models for *continuous* dynamics.

Dynamic Bayesian Network Models. A Dynamic Bayesian Network, or DBN is a form of a Bayesian network designed for modeling dynamical systems. A DBN is a two-layer directed acyclic graph in which each layer represents variables at one point in *time*, and in which all edges are *between* layers[2].

MDPs which use DBNs to represent domain dynamics are referred to as factored MDPs. In a factored MDP, model learning amounts to learning the conditional probability tables of the DBN, rather than the full transition matrix of the MDP. This factorization implicitly marginalizes out all independent variables, which can offer exponential savings in sample complexity. The main difficulty with factored MDPs is that value function does not necessarily factor, even if the model does. Accordingly, much of the research on these models focuses on how to approximate the value function. However, our focus is on model learning, and we will generally resort to online Monte-Carlo methods for approximate value-based planning. Therefore the DBN gives us a starting point for discussing how to exploit structure in dynamics model. As we will see below, despite offering some savings, the DBN

[2]This can be generalized to include intra-layer connections without loss of generality [56]

fails to capture a crucial independence pattern in physical dynamics.

Relocatable-Action Models. The Relocatable-Action MDP (RAMDP) [92] proposed a clustering method for generalizing action effects across states. This was successfully applied to robot-navigation in a small domain without velocity. In each dynamics regime (wood, cloth, or collision), robot motion was sufficiently consistent to cluster together, resulting in a more compact model. The core strength of this approach, in contrast to factored MDPs [33], is that statistical dependency between attributes was no longer stationary but rather depended on a function evaluated at each query state. The OO-MDP [43], discussed next, can be viewed as a successor to this idea, which formalizes the state-clustering process using first-order predicates, and introduces object attributes as arguments to these predicates.

Object Oriented Models. The primary objective behind OO-MDPs is to exploit the unique pattern of stationarity in the dynamics model of tasks with multiple interacting bodies. Unlike previous methods for factoring MDP dynamics, such as Dynamic Bayesian Networks, the OO approach allows model dependencies to vary throughout the state-action space. For example, the next-state of a chair in a kitchen only depends on the state parameters of other chairs if it is about to collide with them. To capture this, an OO-MDP defines a set of object classes $\mathscr{C} = \{C_1, \ldots, C_c\}$ (*e.g.* *table*, *chair*, *wall*), attributes $\mathscr{A}(\mathscr{C}) = \{C.a_1, \ldots, C.a_n\}$ (*e.g.* wheels-locked), and relations $r : C_i \times C_j \rightarrow$ Boolean (*e.g.* *contact-left*$(chair, wall)$) enumerating the possible relationships between objects.

These relations allow dependencies to be formalized as a collection of functions that convert a state into a set of boolean literals, the "condition" associated with that state:

$$Cond(s,a) \mapsto \{p_1(s,a), p_2(s,a), \ldots, p_n(s,a)\}$$

In this fashion, the OO-MDP uses a collection of first-order expressions to partition the state-action space into sets with *homogeneous dynamics*. Model learning then amounts to

learning the action effects under each condition, where a condition is a particular assignment to the OO-MDP predicates and attributes, rather than for each state. The OO-MDP constitutes the closest method from the Reinforcement Learning literature to satisfying the requirements of robot manipulation. However, as we will discuss in greater detail in Chapter 5, this method fails to capture the differential nature of object dynamics.

Non-Parametric Regression Models. Due to their generality, the regression approaches frequently used in RL are non-parametric in nature [34]. Two popular approaches include locally weighted (Bayesian) regression [8] and Gaussian Processes [130]. Locally weighted regression (LWR) will be covered in greater detail in Chapter 5, but we summarize its key properties here. LWR is a "lazy" learning method – rather than fitting a model and discarding the training data, the data is kept and used to evaluate the model by *weighting* instances in proportion to some notion of distance from the query. It therefore maintains many of the favorable properties of linear regression, in terms of analytical tractability and straightforward (least-squares) estimation, but allows for a more general class of functions: locally linear functions. In *Bayesian* LWR, a prior is assigned to the model parameters, which can be incorporated into the estimation machinery to obtain a closed-form expression for the posterior distribution of the one-step prediction given the current state, action, and history of observations [34].

Gaussian Processes (GP) offer a similar "lazy" learning approach but can be analyzed as a prior over functions [130]. In a GP the operations are defined entirely in an inner-product space (the weights in LWR are encoded by a kernel), which permits more abstract prior assumptions such as differentiability or periodicity [34].

The favorable properties of GPs have been explored extensively for model-based and model-free RL, and have led to very efficient model-based policy search algorithms, *e.g.* PILCO [35]. Both LWR and GP assume states and actions can be represented as vectors, and are therefore a good choice for control-level tasks that involve non-linear dynamical systems

(*e.g.* an inverted pendulum). They are useful tools when little is known about the dynamics, but are not sufficient by themselves for the tasks we consider here, which include discrete state variables (*e.g. is-grasped*), and non-smooth dynamical effects (*e.g.* collisions).

2.3 Optimal Control

Optimal control is closely related to model-based Reinforcement Learning, and both fields can trace their roots back to Bellman [12]. Optimal Control is a mature discipline that is applicable to systems in which the dynamics model is a smooth function of state and action (*e.g.* linear or locally-linear).

Similar to model-based RL, these algorithms take as input a dynamics model and a reward (or cost) function that captures some notion of good or bad behavior, and they output a policy (or control strategy) that attempts to minimize this cost. A very common form of optimal control considers linear dynamics and quadratic costs, leading to the so-called Linear Quadratic Regulator (LQR) [14]. These assumptions permit a thorough mathematical analysis of the convergence properties and stability of LQR, but are too strict for many of the problems of interest in this thesis.

Specifically, several of the tasks we consider include discontinuous and nonlinear effects such as wheel friction and object contact that cannot be fully captured by linear dynamics models. In addition, the tasks of interest will have rewards with non-smooth costs that cannot be represented as quadratic forms, *e.g.* the cost to reach a goal pose in a NAMO task Chapter 6 critically depends on whether a collision-free path exists. Lastly, our policy space has discrete combinatorial structure, *e.g.* choosing when and where to grasp an object, or rotating an object while following a trajectory on the ground, that cannot be represented as a linear control law.

2.4 System Identification

The idea of estimating physical parameters from data has a rich history in the robotics, graphics, and computer vision literatures. It arises in vision for model-based tracking [45, 70], and in graphics for data-driven tuning of simulation parameters *e.g.* for cloth simulation [16], rigid-body motion [15], and even humanoid motion [100].

The challenge of controlling an initially unknown system and estimating its relevant parameters online has also been addressed within the controls subfield of *indirect adaptive control* [86]. Adaptation is typically done in two stages. In the first stage, the dynamical system parameters are estimated using a Parameter Adaptation Algorithm (PAA). In the next stage, these parameter estimates are used to update the controller. While most PAA methods assume a linear mode [86], our approach can be seen as an PAA method supporting non-linear model estimation using Bayesian approximate inference. As such, our work is complementary to the existing controls literature.

CHAPTER III

MANIPULATION WITH DATA-DRIVEN MODELS

The goal of this research is to introduce methods for solving real-world manipulation tasks with under-specified models. This is difficult to achieve in practice the underlying state-action space is both continuous and high-dimensional, which poses a challenge for standard value-based dynamic-programming RL methods. However, there has been considerable progress in the planning and control communities on *physical* decision problems [88], albeit typically without considering long-horizon tasks with incrementally gathered world knowledge.

In this chapter we introduce an RL-inspired method that handles continuous and high-dimensional spaces using a combination of established planning and optimization methods in the robotics literature. Like a model-based Reinforcement Learning algorithm, our approach allows the dynamics model to adapt over time as the robot gathers data in the world.

As we will see, this method makes only weak assumptions about dynamics, at the cost of constraining the planner to a set of simple predefined motion primitives. In later chapters we will recover some of the decision-theoretic properties of RL, and generalize our models beyond simple motion primitives. In addition to introducing the problem, this chapter anticipates the general strategy put forward in this thesis: Adding flexibility to physics-inspired methods from the robotics literature by incorporating learning and optimization.

3.1 Planning as Optimization

In the context of service manipulation, the two key advantages of reinforcement learning over classical path planning are that abstract goals can be specified through intuitive rewards and that actions can have uncertain outcomes. Because tasks like cleaning a table require the robot to displace multiple objects, the configuration space for planning has

exponential dimension in the number of objects [158]. This makes it infeasible to apply standard RL strategies. However in many cases we observe that reaching an optimal world configuration is more important than finding the optimal way to reach it. We use this insight to decouple the RL problem into three tasks: (1) determining the goal, or the optimal configuration, (2) finding forward models for robot actions and (3) planning to use the actions to reach the goal.

In this chapter, Sect. 3.1.1 and 3.1.2 describe optimization procedures. Sect. 3.1.3 describes a planning algorithm which seeks a feasible, but not necessarily optimal plan for robot actions. We consider a common service task: cleaning a table. Cleanliness is naturally expressed as an objective function over object poses. We focus on cases in which the objective function is provided by a human programmer. However, as with reward-function learning in RL, these criteria can also be extracted from interactions with humans or the environment. First, we present a method for formulating the objective function. Second, we present a method for learning forward models of object motion. Finally, we combine these elements with sampling-based planning.

3.1.1 Objective Function Specification

Table-setting for an arbitrary number of guests is an abstract goal. This goal is fundamentally distinct from positioning plates at desired locations since it is the spacing between the dinnerware that matters to the guests rather than their precise locations. Using our method, the programmer can specify the goal as an abstract optimization metric.

Without loss of generality, consider a dinner where n guests must be given n plates and m platters must be placed at the center of the table. The programmer should be able to state the following objectives:

1. The plates should be located far from each other.
2. The platters should be at the center of the table.
3. The platters should be aligned parallel to the table.

The same plate positions will not satisfy these criteria for different numbers of guests. However, the optimization criteria indicated by the objectives are easily formulated with Eq. 1-3. We define two sets of objects: plates, P, and platters, Q. Each object location is parameterized by position and orientation $\{x, y, \theta\}$. For a rectangular table, parallel to the frame of reference its center is (x_G, y_G).

$$c_{dist} = -\sum_{p1 \in P} \sum_{p2 \in P} \sqrt{(x_{p1} - x_{p2})^2 + (y_{p1} - y_{p2})^2} \quad (1)$$

$$c_{pos} = \sum_{q \in Q} \sqrt{(x_q - x_G)^2 + (y_q - y_G)^2} \quad (2)$$

$$c_{ortho} = \sum_{q \in Q} |(\text{mod}(\theta_q, \frac{\pi}{2})| \quad (3)$$

The programmer or high-level planner should also specify environment constraints. For example, Eq. 4 limits all objects to the table dimensions:

$$x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}, \quad (4)$$

More generally applicable optimization criteria can also be specified. For instance, the table center (x_G, y_G) can be inferred from the table dimensions as shown in Eq. 5.

$$x_G = \text{argmax}_x(\min(x - x_{min}), (x_{max} - x))$$
$$y_G = \text{argmax}_y(\min(y - y_{min}), (y_{max} - y)) \quad (5)$$

The overall objective for cleaning a table is simply the sum of our intuitive criteria as given in Eq. 6. The weights α, β, γ must be specified with regard to the relative importance of the subtasks.

$$C_{table} = \alpha c_{dist} + \beta c_{pos} + \gamma c_{ortho} \quad (6)$$

3.1.2 Action Model Learning

The second optimization problem we consider is determining the relationship between robot actions and their effects on objects. Here we consider only collision free motions,

and handle collision-avoidance in the planner. This approach requires significantly less data than when modeling all possible contacts between an unbounded number of objects. We now describe the action model learning procedure which allows the planner to reason about the outcomes of its actions in various states.

In our implementation, the robot was equipped with a ball-shaped end-effector that could only push objects. In other domains, the robot might grasp objects. In either case, a service robot will encounter new objects. Different pushes or grasps will have distinct effects on object displacement. For simplicity, consider the pushing domain. Our actions change the pose of an object, O, placed on the table at $\{x_O, y_O, \theta_O\}$.

In order to encode uncertainty in action outcomes we follow the definition of Markov Decision Processes (MDP). Our action model is a function that maps state and action to a probability distribution over states. Given some state-space S and actions A, a planner must know the probability of a given outcome of any action in any state:

$$P(s'|a,s) \; \forall \; s \in S, \; a \in A \tag{7}$$

Our approach automatically generates $P(s'|a,s)$ from data obtained by exploration. In our domain, object displacement has a direct relationship to the motion of the end-effector relative to the object rather than relative to the world. We defined six actions relative to the object as shown in Fig. 4. We then compute *probability models of displacement*:

$$P(\Delta s|a,o) \; \forall o \in O, \; a \in A \tag{8}$$

To estimate $P(\Delta s|a,o)$ for every action applied to every object, we implemented the optimization algorithm shown in Fig. 1. LEARN_MODEL incrementally completes a tabular probability distribution that relates each action, $a \in \mathbf{A}$, to the displacement, Δs, for the object, $o \in O$. On each iteration, the robot applies an action to the object, observes the result using an overhead camera, and uses UPDATEDISTR to update the probability distribution, $P(\Delta s|a,o)$, as shown in Eqs. 9-10. Our model represents the probability of displacement

```
LEARN_MODEL(o, A, s_init)
1  s = s_init
2  for i ← 1 to |A|
3  do while σ² > σ²_ref
4     do (Δs, s) = APPLY_ACTION(a_i, O)
5        P(Δs|a_i, o) = UPDATEDISTR(P(Δs|a_i, o), a_i, Δs)
6        σ² ← VARIANCE(P(Δs|a_i, o))
```

Figure 1: Pseudo-code for learning forward models. Note that Δs encodes state transitions in object coordinates.

with a normal distribution. This requires UPDATEDISTR to update the distribution mean and variance. For each variable in state $\mathbf{s}$, $\{v = x, y \text{ or } \theta\}$, the n^{th} update is given as follows:

$$\mu_v^n = \frac{n\mu_v^{n-1} + \Delta v}{n} \tag{9}$$

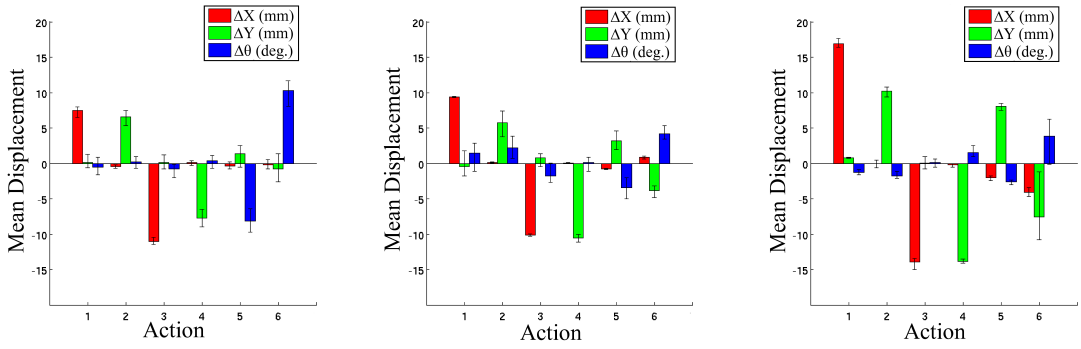
$$\sigma_v^n = \sqrt{\frac{(n-1)\,\sigma_v^{2(n-1)} + (\Delta v - \mu_v^n)\,(\Delta v - \mu_v^{n-1})}{n}} \tag{10}$$

Iteration terminates when the variance of the distribution reaches a desired threshold, σ_{ref}^2. This criterion identifies that the model is sufficiently accurate for planning. In our experiments, we empirically determined that $\sigma_{ref}^2 = 0.4$ yielded a reasonable compromise between execution times for learning and plan execution for our table-top domain.

The model achieved by the algorithm for three objects is illustrated in Fig. 2. The figure shows mean displacements and confidence intervals when the robot applies our six actions to each object. Notice the significant variation in these parameters. This is precisely the reason that learning object models is essential to the construction of valid plans.

3.1.3 Task-Space Planning

Given a task-level goal and a forward model that relates robot actions to world effects, the remaining challenge is to produce a planner that efficiently finds a sequence of actions that achieves an optimal configuration. In order for this optimization process to be useful



(a) Eraser (b) Remote (c) Scale

Figure 2: Model learning results with three objects: a chalkboard eraser, a TV remote, and a digital scale. Bars show the mean and 95% confidence intervals extracted from each object for all degrees of freedom. Actions are those illustrated in Fig. 4.

it must respect the physical limitations of the environment/robot and be reachable from the initial state of the problem. The first condition is satisfied by imposing collision and boundary constraints (Eq. 4). To satisfy the second, the optimizer must also respect the motion constraints specified by our models.

We present a motion planning algorithm that directly searches the optimization landscape using models of the robot's actions. By combining optimization and motion planning into a single search, we restrict the search to states that are relevant given the robot's available actions. Our algorithm prevents the planner from aiming towards optimal configurations that are unreachable. Furthermore, since the planner always knows the best state in its search graph, it offers more reliable anytime characteristics than approaches that separate goal optimization from planning. The remainder of this section describes the basic functionality of the algorithm, as well as an extension which makes it more efficient when searching large spaces.

Our algorithm, given in Fig. 3, extends RRT [81, 89], a probabilistically complete method that is commonly applied to motion planning in high-dimensional spaces. Basic RRTs produce a tree of valid, collision-free configurations for n-dimensional spaces through incremental expansions of the tree towards random configurations. Application of the basic RRT to problems such as manipulator control relies on some form of inverse

```
TASK_SPACE_RRT(s_init, A)
 1  for i ← 1 to |O|
 2  do Model ← LEARN_MODEL(O_i, A, s_init)
 3  T.init(s_init)
 4  for i ← 1 to max_nodes
 5  do s_GD ← DIRECTGD(T)
 6     if RAND() > ε
 7       then s_samp ← s_GD
 8       else s_samp ← RANDOM_CONFIG()
 9     s_near ← NEAREST_NEIGHBOR(s_samp)
10     a* ← ARGMIN_a(ρ (MODEL(s_near, a), s_samp))
11     s_new ← MODEL(s_near, a*)
12     if not IN_COLLISION(s_new)
13       then ADD_VERTEX(s_new)
14            ADD_EDGE(s_near --a*--> s_new)
```

Figure 3: Pseudo-code for RRT-based task space planner using forward models. T is the search tree, and ρ is the distance function defined in Eq. 11.

model for the system to determine how to reach a child node from its parent. When working in task space, however, we have no clear model for how to transition between nodes. Instead we modify our approach to perform a forward search over the robot's possible actions. The core idea is that by confining the search to a set of pre-defined actions for which we have controllers, we can both restrict the search to reachable states, and relieve the need for an inverse model. A similar approach is given by Frazzoli [49].

Our method preserves the *rapidly exploring* characteristics standard RRT search with two modifications. First, rather than growing the tree directly towards sampled states, we select the state-action pair which results in a node closest to a sample point according to a weighted distance metric. If this state is valid, we add a directed edge labeled with the appropriate action to the tree. Since the best action can fail the subsequent collision check, we maintain state-action pairs in a queue ordered by distance. The planner iterates through this list and adds the first collision-free pair that is closer than the parent node. This modification yields efficient extensions to new states using only forward models.

The second unique modification is the addition of a conventional gradient descent optimization routine, DIRECTGD, during the planning process. DIRECTGD is a heuristic that searches the objective function directly, without branching over robot actions. Periodically executing this search from leaf nodes in the tree provides a way to quickly discover the nearby local minima of the objective function. Empirically, we found that this heuristic significantly speeds up search towards the global minimum while the random action of the RRT prevents the overall planner from getting stuck in any local minimum. Our technique is inspired by existing RRT heuristics like RRT-CONNECT [81] which try to extend the tree towards a goal or a goal tree. However, in our case the "goals" are the modes of the optimization landscape, and DIRECTGD allows us to explore this space more efficiently. If DIRECTGD returns a more optimal state than one that currently exists in the planner's search tree, it attempts to apply RRT-CONNECT using the robot's actions to find a path to this state.

As with a conventional RRT, our planner contains an ε parameter that trades exploration and exploitation. The planner takes greedy steps towards the lowest-cost state identified by DIRECTGD with probability $1-\varepsilon$. In our domain we empirically determined that $\varepsilon = 0.8$ gave the fastest convergence to a global optimum.

In addition to the two RRT algorithm modifications we must design a distance metric that accurately reflects the similarity between states:

$$\rho : f(s_1, s_2) \rightarrow \mathbb{R} \tag{11}$$

This distance function is used to query nearest neighbors in TASK_SPACE_RRT (line 9). In task space, the parameters used to describe state are specified by the programmer. They are not required to share a coordinate frame or scale. For our table-top manipulation experiments we empirically determined that weighing orientation parameters $40:1$ over position produced the most consistent alignment of objects.

3.1.4 Executing Motion Primitives

We designed a set of six simple motion primitives for 2D manipulation with a single-point contact end effector (Fig. 4). In order to minimize the branching factor of the planner, we defined actions based on relative motions with respect to simplified object geometry.

The robot identified objects in its workspace using a simple threshold-based segmentation scheme with an overhead camera. Objects were extracted by finding closed contours in a mask image. Each object was represented by a bounding box aligned with its long axis and then classified based on the size and aspect ratio of the bounding box. The position and orientation parameters were taken as the centroid and the angle of the box from horizontal, respectively. We defined six primitives to control the three degrees of freedom as independently as possible, one for increasing and one for decreasing each parameter.

Each of the six primitives was defined by a workspace vector in the object frame (Fig. 4). During action execution, the robot lifted and lowered its end-effector to the point on the bounding box along the vector direction. It then pushed the object by tracking the vector in workspace for a distance of 5cm. The robot transformed the nominal workspace trajectory into the arm's joint-space with analytical inverse kinematics and tracked the reference with PD control. In order to produce mostly linear translations of the object, actions 1-4 were set as vectors normal to the bounding box at the midpoint of each face. Actions 5-6 were set as the two opposing vectors at 90% of the box length along the long-axis box faces, targeting rotation.

The effect of primitive actions on each of three classes of objects is given by Fig. 2. During learning, we found that each action's influence was largely confined to the state parameters it was designed to influence. However, the extent of influence varied greatly due to differences in physical properties. For the scale, rotation actions also produced significant changes in position terms. This was due to the rubber feet on the corners of the scale, which tended to break loose and skip all at once as the robot pushed from the end. One advantage of this approach is that observable dynamics which are difficult to model

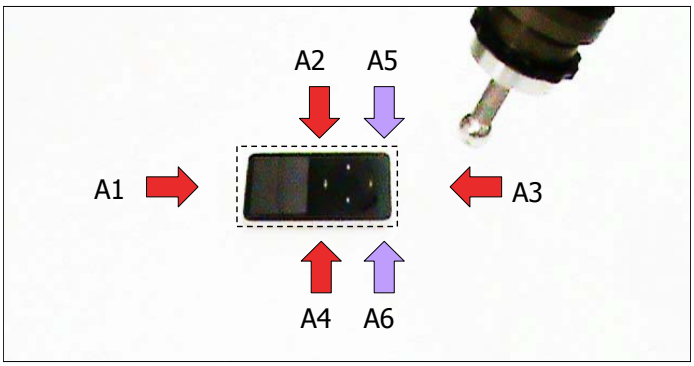


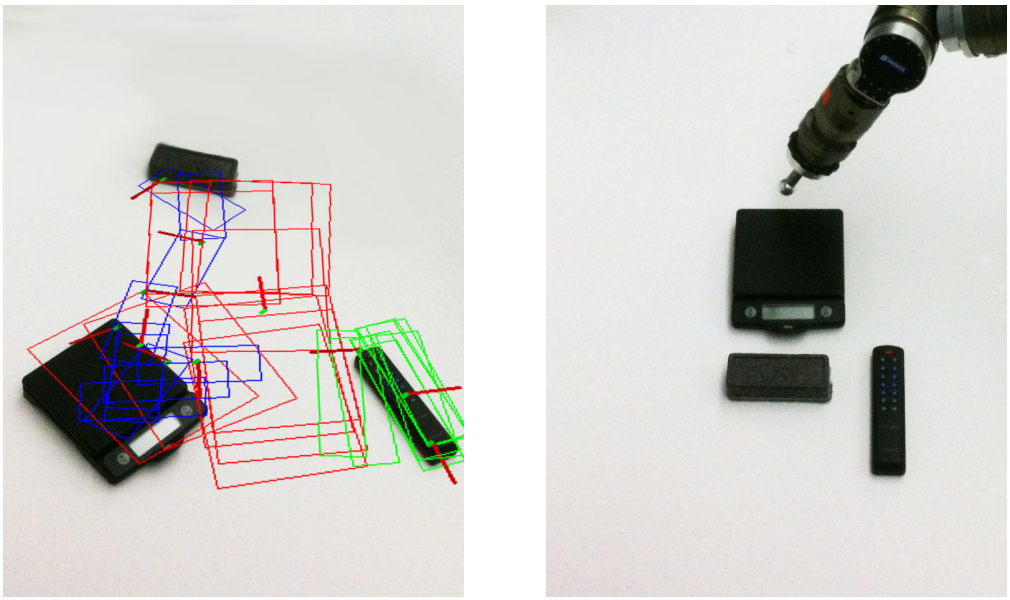
Figure 4: Six motion primitives defined with respect to a bounding box around a sample object. Arrows indicate motion vectors for the end-effector as defined in the object frame.

explicitly are captured through learning and accounted for at plan time.

3.2 Experiments

We conducted four experiments using the TASK_SPACE_RRT planner. The first two experiments were simulated table-setting tasks. These were performed in SRLib, which replicated a real world task domain with dynamics and visually perceived object states [121]. We used the SRLib block and cylinder primitive shapes to represent plates, platters, and a table. In both tasks the problem was to optimize the table-setting function (Eq. 6). Fig. 6 and 7 present two manipulation plans from our algorithm that generate neat tabletop configurations from random configurations of three and six objects. Notice that the planner gracefully adapts to object quantity and geometry resulting in different final configurations under different circumstances. Fig. 10 gives the planning statistics and shows the performance improvement from the GD heuristic. The greatest challenge in larger problems was collision detection when the free-space became increasingly taken up by objects. Most RRT steps, both greedy and random, fail when the planner becomes deadlocked, reducing the algorithm to a slow random search.

For the first demonstration on the physical robot we designed a simple task of positioning an object in the presence of a stationary obstacle. We reused the distance (Eq. 2) and orthogonality (Eq. 3) terms from the previous experiment, and added an additional

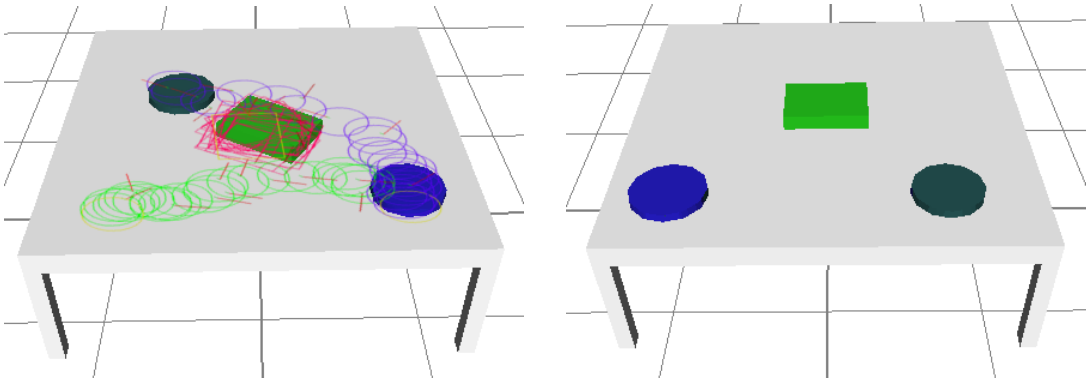


(a) Initial State, Plan (b) Final State

Figure 5: Table configurations before and after executing a plan computed to optimize the table-cleaning objective function. Colored outlines depict the sequence of states visited during execution for each object.

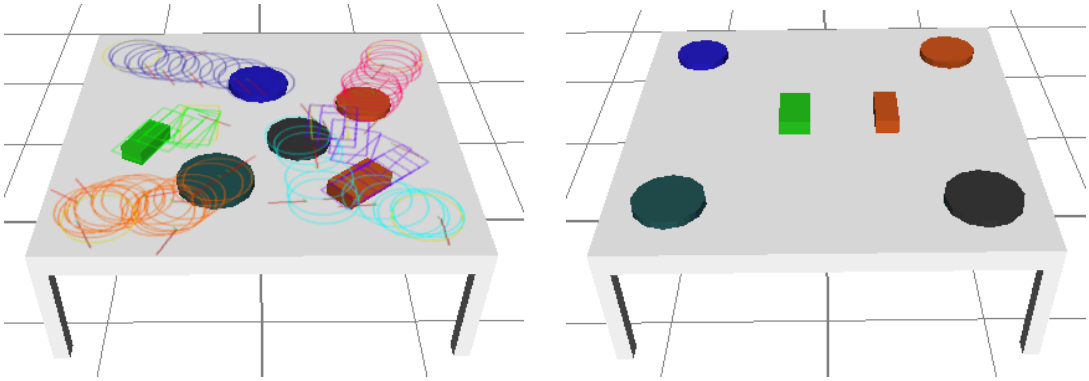
constraint in the form of a fixed obstacle. The eraser was used for the mobile object, and the hole-punch was labeled as an obstacle (Fig. 8). After learning a motion model for the eraser, the robot found a plan that involved pushing the object up and around the obstacle to a goal point on the left side of the table (Fig. 8).

To demonstrate a more practical application, the goal our final experiment was to make the robot straighten up a desk. The scale, the remote, and the eraser were distributed randomly around the workspace at the start of the experiment, and it was the robot's job to find a sequence of actions which minimized an objective function for "table-cleaning" with the same form as (Eq. 6). The planner execution for this task can be seen in Fig. 5. After learning models for the three objects, the robot proceeded to push the three objects into orthonormal alignment with a center of mass less than two inches from the center of the table. Execution took 13 minutes, and required re-planning 4 times (Fig. 9).



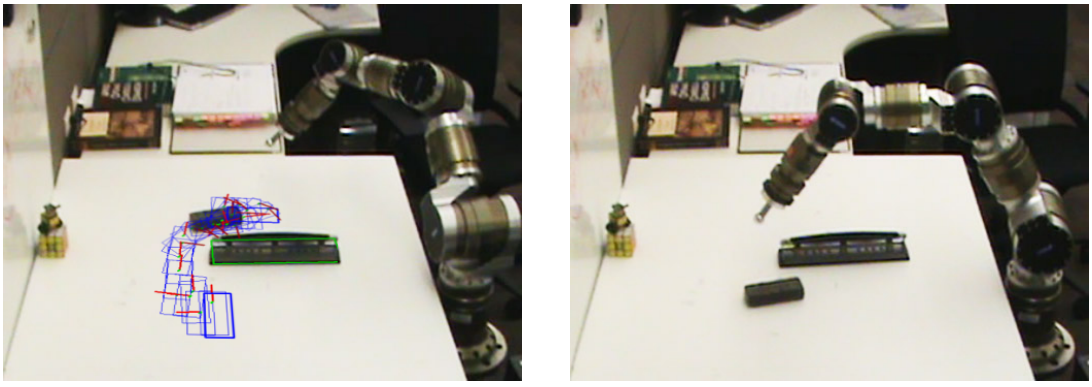
(a) Initial Plan (b) Result

Figure 6: Execution of the table-setting planner in the srLib simulator with two plates and one platter. Colored outlines depict the sequence of states visited during execution for each object.



(a) Initial Plan (b) Result

Figure 7: Execution of the table-setting planner in the srLib simulator with four plates and two platters. Colored outlines depict the sequence of states visited during execution for each object.



(a) Initial Plan (b) Correcting from error

Figure 8: Manipulation in the presence of obstacles

	TSet 6	TSet 3	Obst	Clean
n-nodes w/ *GD*	2161	1438	541	981
n-nodes w/o *GD*	1898	1399	549	954
time (m:s) w/ *GD*	0:32	0:18	7:22	13:40
time (m:s) w/o *GD*	1:34	0:33		
n-steps w/ *GD*	114	122	44	89

Figure 9: Selected statistics on planning for 6 object table-setting, 3 object table-setting, 1 object obstacle, and 3 object cleaning.

Task	TSet 6	TSet 3	Obst	Clean
DOF	18	9	3	9
# replans	1	1	2	4

Figure 10: Degrees of freedom (DOF) and the number of required re-plans for each problem in Fig. 9

3.3 Discussion

In this chapter we presented an algorithm for cost-based task planning with empirical dynamics models, using a novel combination of sample-based motion planning and optimization. While not *physics-based* in the same sense as the *PBRL* method introduced in Chapter 5, there are several physical ideas implicit in TASK_SPACE_RRT. The first is *probability models of displacement*, an idea we will extend with *Object-Oriented Locally-Weighted Regression* Chapter 5. These displacement models are discrete approximations of the true differential dynamics of each object, but lacking force or velocity terms. Furthermore, using them requires mapping the sampled displacement to the frame of the query node using a homogeneous transform (an operation specific to 3D euclidean space). Second, TASK_SPACE_RRT makes use of a collision detector to avoid pushing objects into contact with one another. While this doesn't capture full collision dynamics like the physics-engine in Chapter 5, the collision detector does require geometry-based computations.

From these results we conclude that sample-based planning and empirical displacement models are a potent combination for multi-object planning in physical environments. Such an approach leverages the well-known insight from Reinforcement Learning that it is often faster to learn a model to use for online planning than it is to learn a full policy.

However, there are several shortcomings of TASK_SPACE_RRT that limit its applicability to problems of interest, such as NAMO. Foremost, the actions were open-loop pushes that failed to exploit grasp capabilities or take the robot's body into account. In addition, motion primitives provide a very restrictive action space for object manipulation. Sequencing primitives is a slow and brute-force method of search, and the planner therefore required a separate optimization module to explore efficiently. Incorporating the robot's body and grasp kinematics will be the topic of the next chapter, and we leave it to Chapter 6 to enrich the action space to achieve arbitrary directions.

CHAPTER IV

PLANNING WITH KINEMATIC CONSTRAINTS

The previous chapter contributed a novel algorithm for object manipulation that offered some of the flexibility of the Reinforcement Learning framework, but could handle continuous high-dimensional state spaces. Although this represents a major step towards our goal, the planner relied on fairly crude primitives which drastically limited the set of achievable behaviors. Most importantly, these primitives were designed for tabletop manipulation with an overhead arm, and therefore failed to consider the geometric or dynamic constraints of the robot itself.

Full-scale mobile manipulation also introduces additional forms of complexity due to the need to localize and safely maneuver a high degree-of-freedom (DOF) robot in a large workspace, possibly including dynamic obstacles such as people. As a result, the method in Chapter 3 is not directly applicable to *mobile* manipulation tasks.

In this chapter we address these limitations by incorporating the components necessary to achieve full mobile manipulation planning with a humanoid robot. We present a planner for *articulated* mobile manipulation of a single holonomic object with known kinematics and geometry. This chapter introduces a one degree-of-freedom deformable collision model, parameterized by grasp angle, which represents a system including both the robot and a rigidly-grasped object. We will show that this model increases the manipulation capabilities of the robot, and is critical for navigating narrow spaces. As we discuss in Chapter 6, these abilities also lay the foundation for more general manipulation strategies with *constrained* objects. This work was originally published in [142].

4.1 Motivation

The earliest results in pushing carts using robots were achieved using a single manipulator mounted on a mobile holonomic base [169–171]. In these systems the manipulator came into contact with the cart at a single point, and the problem was to solve for the effector forces required to produce desired trajectories with the cart. This work showed progress towards task-level cart manipulation, but was limited to tracing simple open-loop paths with the cart [171]. In contrast, our solution can execute arbitrary smooth trajectories in a closed-loop controller using two arms, assuming the object is holonomic.

Subsequent work explored the use of full humanoid robots for pushing carts with two arms. In principle, two arms can simplify the cart control problem by fully constraining all degrees of freedom. Honda's ASIMO is capable of pushing a cart while walking across a room and even up an incline, but does not make use of its arms to articulate the cart for navigation [148].

Several projects have explored the use of another humanoid robot, the HRP-2, for pushing mobile objects. One domain which shares our interest in manipulating large objects is *navigation among moveable obstacles* (NAMO), which we will discuss in depth in Chapter 6. Although both domains involve navigation and manipulation, NAMO addresses the complementary problem of navigating the robot on a map containing obstacles that must be moved in order to reach a *robot* goal pose [105]. Rather than focusing on navigation *among* static objects, this work focuses on navigation *of* objects, amidst other dynamic obstacles.

In another mobile manipulation application with a biped humanoid, [120] presented an HRP-2 capable of pushing a human in a wheelchair. As with ASIMO and NAMO, this project described a zero-moment-point (ZMP) offset approach to achieving basic mobility with a mobile object [69]. However, none of these examples with humanoid robots demonstrated robust navigation in dynamic, cluttered environments with tight clearances for the robot, and none took advantage of both arms for articulation. The PR2 mobile manipulation robot used in this study offers a stable, omni-directional base. Its ability to articulate

the cart greatly enhances the reachable workspace for the robot since it can now take tight turns in cluttered spaces.

The work that is closest to ours is the approach of Lamiraux, et. al., who demonstrated motion planning for a robot towing a trailer [84]. The trailer is attached to the robot through a single degree of freedom pivot joint. In our approach, we restrict the motion of the cart to be around an (imaginary) pivot point in front of the robot. In contrast to [84], we present a complete solution integrating 3D sensing to develop a system capable of realtime navigation in a cluttered indoor environment.

4.2 The PR2 Hardware Platform

The hardware used for the experiments in this chapter is the PR2 personal robot (Figure 11) which has an omni-directional base and two 7-DOF arms. It is also equipped with a tilting laser scanner mounted to the head, two stereo cameras, an additional laser scanner mounted on the base, and a body-mounted IMU. Encoders on each joint provide joint angle information. The end-effector is a parallel jaw gripper whose fingertips are equipped with capacitive sensor arrays, each consisting of 22 individual cells. The laser scanner mounted on the base is useful both for obstacle detection and localization. The robot's base is approximately 63 cm in both length and width.

The cart used in the experiments in this work is a regular holonomic utility cart. It has casters mounted at all four corners and can thus be pushed in any direction. The top shelf of the cart was removed to reduce the volume of the region in front of the cart that is occluded from the PR2's tilt scanning laser sensor mounted on the head. The PR2 grasps the handle of the cart as shown in Figure 12(a). The grasp is sufficiently rigid to maintain its relative pose with respect to the cart handle.

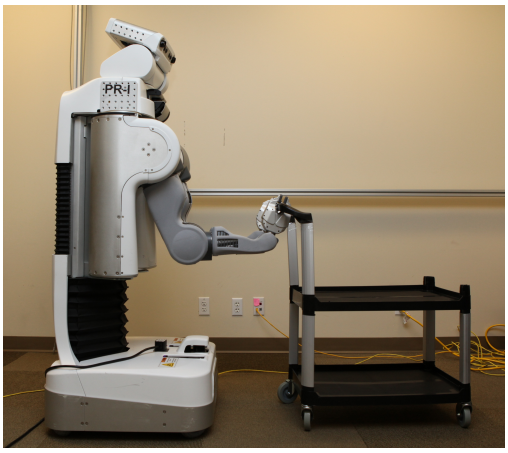
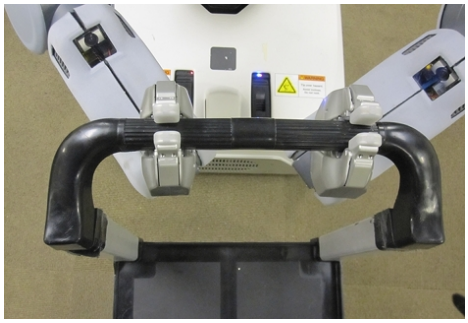
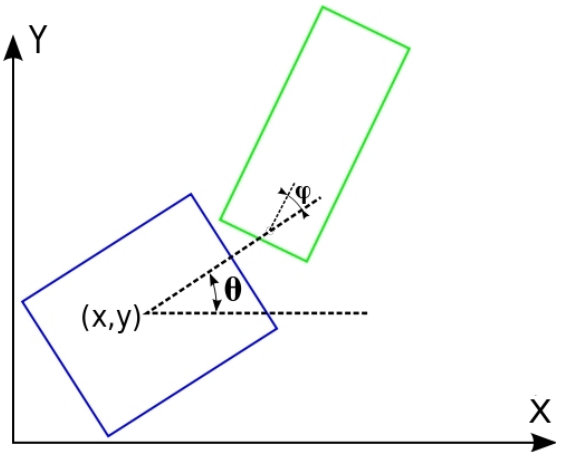


Figure 11: The PR2 mobile manipulation robot with a holonomic cart



(a) The PR2 has a firm grasp on the cart handle (right).



(b) State representation: Blue rectangle represents the robot, and θ its orientation. Green rectangle represents the cart, and ψ its orientation in the robot frame.

Figure 12: Cart grasping (a), and state representation (b)

4.3 Approach

Navigation while manipulating moveable objects is a challenging planning and control problem. Ideally, a planner used for this task should efficiently take advantage of the latent capabilities of the robot, and react quickly to failure given a cluttered and dynamic environment. We make almost no assumptions about the structure of the environment. For simplicity, we assume a known 2D map of the environment as a starting representation. The map was built separately from real sensor data using tools available in the ROS framework [129]. This map is not, however, assumed to be a completely accurate representation of the environment, which can contain both static and dynamic obstacles such as people or other robots. We demonstrate, through experiments, the ability of our approach to deal with such static and dynamic obstacles that are not initially known to the robot. We assume a known geometric and kinematic model for the cart. This helps us in localizing and controlling the cart relative to the robot. The task of determining a dynamic model for the cart is left to Chapter 5.

Overall our approach consists of a tight integration of three different components: sensing for the cart and the environment, motion planning for the robot and the cart, and control of the robot and the cart along the desired path.

4.4 Sensing

Our approach to mobile manipulation builds on components developed for navigation and manipulation with the PR2 [106, 135]. To navigate effectively, our system must be able to differentiate between sensor readings that may correspond to points on the cart or the robot, and those from points in the environment. Sensor readings corresponding to points on the cart are filtered out directly from a 2D costmap representation (Figure 13) of the collision environment if their 2D projection falls within the known polygonal footprint of the cart.

Two approaches were implemented to sense the pose of the cart relative to the robot. First, a checkerboard attached to the cart was used to localize the cart pose relative to

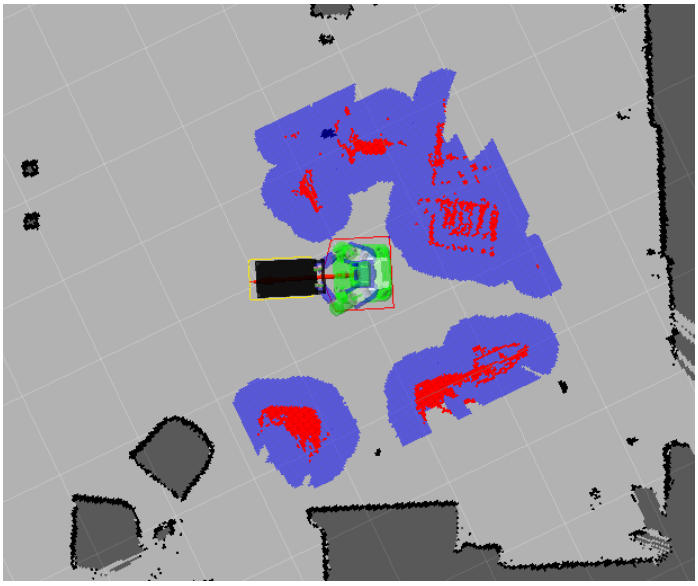


Figure 13: A view of the local costmap. Red cells are lethal obstacles while blue cells lie within a threshold distance of the nearest obstacle.

the robot using the cameras mounted on the head of the PR2. Second, the known initial positions of the grasps of the two end-effectors on the cart were used to provide a *proprioceptive* estimate of the cart relative pose. Note that this assumes that the cart handle stays rigidly fixed relative to the end-effectors of the two arms, which proved a safe assumption in our experiments. Indeed, we found the proprioceptive estimate to be more stable than the checkerboard estimate due to the visual noise and jitter in the camera.

4.5 Motion Planning

The motion planner provides global collision-free plans between the start pose (the current pose of the robot) and the final desired position of the robot and the cart. There are several approaches to motion planning for mobile robots that could be applicable in this case, including both graph and sampling-based planners [83]. We choose to use a graph based approach coupled with an anytime planner. A graph based method was selected for this task because the configuration space is dominated by numerous long, narrow passages (see Fig. 15), which are pose a greater challenge for sample-based planners. As we shall

describe in the next few sections, the choice of such a planner allows us to specify candidate motions of the robot and cart system (motion primitives) that allow the robot to navigate tight turns.

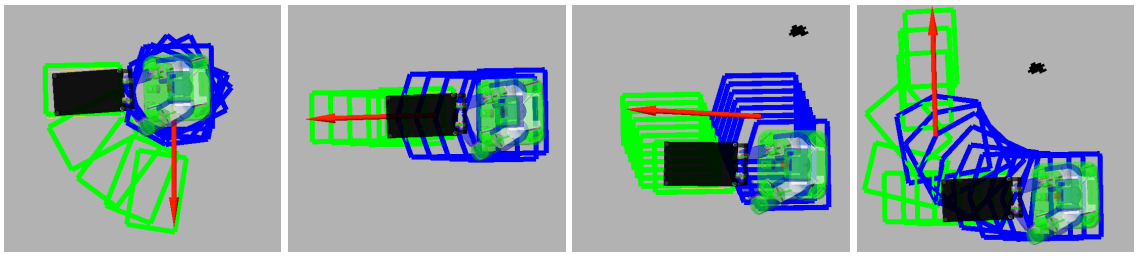
We first describe the construction of the graph itself and then we describe the construction of the transitions between the nodes in the graph. We will then briefly describe the algorithm used to search the graph for low-cost solutions.

4.5.1 State Representation

To construct the graph, we need to specify the state representation for the nodes of the graph. A full state representation would include the 7 degrees of freedom for each arm, the 3 degrees of freedom of the robot base and the 3 planar degrees of freedom of the cart relative to the robot base. However, as noted earlier, the arms of the robot are constrained by the grasp on the cart, which itself is constrained to planar motion. One simpler representation to eliminate redundancies could be the planar degrees of freedom of both the base of the robot and the cart (in the robot reference frame). Such a representation $(x, y, \theta, x_c, y_c, \theta_c)$ would result in a highly controllable 6-dimensional state space for the system. However, we choose instead to restrict the motion of the cart by specifying a fixed point of articulation for the cart in the robot frame. This choice is motivated planning considerations — it reduces the dimensionality of the search space for planning while still retaining enough flexibility to allow the articulation needed to execute tight turns. Our choice of state representation (x, y, θ, ϕ) is shown in Figure 12(b).

4.5.2 Transitions

The transitions between nodes in the search graph are defined using a lattice-based planning representation [97, 125]. A lattice-based planning representation discretizes the configuration space into a set of states. The connections between the states are also discretized and every connection represents a feasible path. The lattice representation can be used to



(a) (b) (c) (d)

Figure 14: Four example motion primitives. The red arrow indicates the final pose of the robot at the end of each primitive. (a) rotate in place, (b) move forward, (c) move diagonally, (d) articulated right-turn.

specify the motion planning problem as a graph-search. The key advantage of this representation, in contrast to other approaches like 4-connected or 8-connected grids, is that every connection between states is a feasible connection. This makes the lattice-based representation a good choice for constrained systems, such as a robot with an articulable cart.

Since the PR2 robot is omni-directional, we choose to enable transitions that allow the robot to move forwards, along diagonal paths, rotate in place, move backwards and move forwards and backwards while rotating. These transitions are thus *motion primitives* and can be used to generate a search graph of the task space. Note that these primitives differ from those used in Chapter 3 in that they are executed by a closed-loop manipulation controller with prehensile grasp, and therefore present no uncertainty to the planning system. In order to exploit the controllable degrees of freedom of the robot+cart system, we also designed primitives for simultaneous navigation and articulation. Four of the motion primitives used are illustrated in Figure 14. During the search process, the planner checks the entire motion primitive between parent and child nodes for collisions using the global costmap.

4.5.3 Cost Function

In general, the planner can accommodate an arbitrary cost function reflecting such objectives as travel time, risks involved in hard-to-execute maneuvers, distance from obstacles and other metrics. In our experiments, the cost of each edge in the constructed graph corresponded to the time required to execute the motion primitive represented by the edge. The cost of the edge was also increased further when the corresponding motion primitive traveled close to an obstacle. Finally, we also increased the cost of edges for certain motion primitives that were harder to execute.

4.5.4 Graph Search

Given a graph defined as above and a cost function associated with each action, an efficient search method is required for finding a solution path. A* search is one of the most popular methods for this problem [58]. It utilizes a heuristic to focus the search towards the most promising areas of the search space. While highly efficient, A* aims to find an optimal path which may not be feasible given time constraints and the dimensionality of the problem. To cope with limited deliberation time, we use an anytime variant of A* — Anytime Repairing A* (ARA*) [98]. This algorithm generates an initial, possibly suboptimal solution quickly and then concentrates on improving this solution while time allows. The algorithm guarantees completeness (for a given graph) and provides bounds on the sub-optimality of the solution at any point of time during the search. Furthermore, this bound, denoted by ε, can be controlled by a user. In all of our experiments, we set the initial ε to 3.0, implying that the cost of the returned solution can be no worse than 3.0 times the cost of an optimal solution (even though the optimal solution is not known). In nearly all of our experiments, ARA* was able to decrease the bound on on sub-optimality to 1.0 (corresponding to a provably optimal solution) within the time we allocated for planning.

Similar to [97], the heuristics we used were computed online as costs of 2D (x, y) paths that take into account obstacles. These heuristics were computed via a single Dijkstra's

search before each planning episode.

4.5.5 Local Control

Given a path from the global planner, i.e., a set of waypoints of the form (x, y, θ, ϕ), a *local controller* is responsible for translating this path into commands for the base and arms. It runs at 20 hz in the odometric (rather than the map) frame. It therefore provides robustness against dynamic obstacles and localization errors. It aims to follow the global plan as closely as possible: if the plan is found to be in collision, it will slow down or stop rather than attempt to move around the obstacle. If it is unable to follow the global plan, the local controller aborts, forcing the system to re-plan a global path.

The local controller operates by moving its desired goal along the global plan returned by the planner. Desired base velocity (v_b) and cart (v_c) velocities are determined using a proportional controller (Equation 12).

$$v_b = T_b K_p^b e_b \tag{12}$$

$$v_c = K_p^c e_c \tag{13}$$

where e_b and e_c represent the errors in base pose (expressed in the global frame) and cart pose (expressed in the robot frame), T_b transforms the desired base velocity into the base frame and K_p^b and K_p^c are positive definite gain matrices. The errors are given by,

$$e_b = q_b^d - q_b^a$$

$$e_c = q_c^d - q_c^a$$

The superscripts d and a indicate desired and actual velocities respectively. The desired base and cart command velocities are scaled appropriately based on desired maximum speeds for the base and the cart.

Note that because of our choice to parametrize the base motion by articulating the cart about a fixed point in the robot base frame, the position of the center of the cart and orientation of the cart $q_c = (x_c, y_c, \theta_c)$ are a function of the cart articulation angle ϕ. The position

and orientation of the base in the global frame, $q_b = (x, y, \theta)$ is completely specified by the plan. The desired base (v_b) and cart (v_c) velocities are determined using a proportional controller.

The velocities for the base and the cart are forward simulated in the odometric frame to check for collisions against the local costmap. If they result in collision, the velocities are scaled down until a collision is avoided, and failure is declared by the local controller if no scaling is possible. The failure of the controller triggers global re-planning. These steps serve to slow the system down when operating near obstacles, and results in re-planning if failure of the local controller is unavoidable.

4.5.6 Cart Articulation

Articulation of the cart is critical for narrow-tolerance navigation problems, such as passing through doors, rounding corners in narrow hallways, or avoiding obstacles in densely cluttered areas. Where previous approaches required error-prone analysis of contact forces and dynamics, the presence of two arms allowed us to produce accurate cart trajectories simply by transforming the desired cart velocity appropriately into desired velocities for the two end-effectors (v_{ee}). These velocities are then mapped into the joint space of each arm using the transpose of the manipulator Jacobian. The net result is a set of desired joint velocities (q^d_{arm}) for each arm that allow the system to track the desired cart velocity:

$$\dot{q}^d_{arm} = J^T v_{ee} \tag{14}$$

4.6 Experimental Results

We conducted extensive experimentation to validate the robustness and reliability of our system. The experiments were carried out in a typical office building with furniture and people. The robot was completely autonomous and would follow a sequence of waypoints defined on the building map. Waypoints were chosen such that the robot would have to

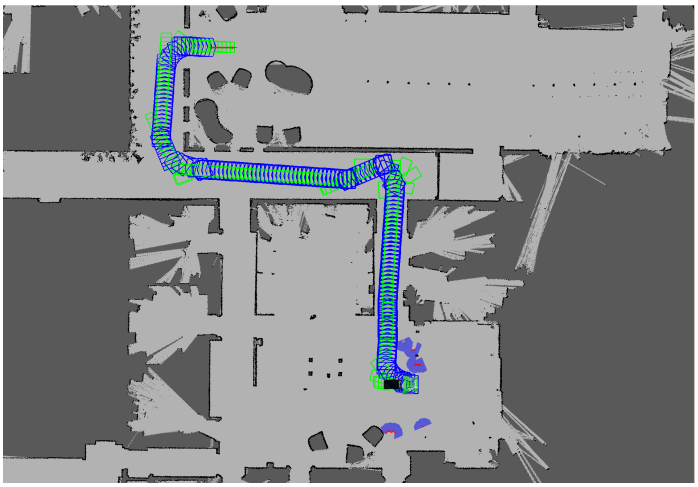
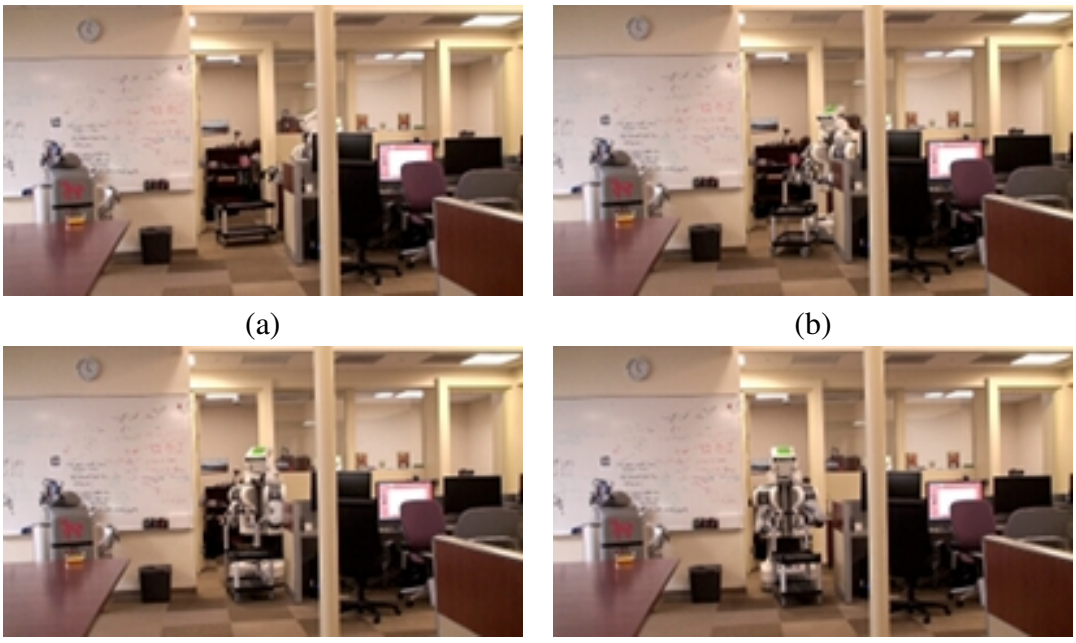


Figure 15: A long planned path using the lattice-based planner. The red arrow indicates the goal for the robot base, the blue polygon is the footprint of the robot during the plan and the green rectangle represents the cart. Note the frequent use of the articulated motion primitives.

move through tight spaces and high-traffic areas. The global planner was allocated a maximum planning time of 10 seconds. In most cases, the planner took significantly less time to plan the first solution. Data logged for the global planner included time to first plan, time to final plan, and cost of the plan. An example global plan returned by the planner on the global map is shown in Figure 15 which shows the plan for a distant goal.

The system performed exceptionally well. It was able to handle the presence of people in its vicinity, stopping when they suddenly stepped in front of it, and either re-planning or restarting once they moved away. It never hit any obstacle in the environment. Figure 16 provides a series of snapshots of a run of the robot as it was performing a tight 90 degree turn. Figure 17 contains a series of snapshots showing the robot stopping in time to avoid hitting a person and then planning a path around the person.

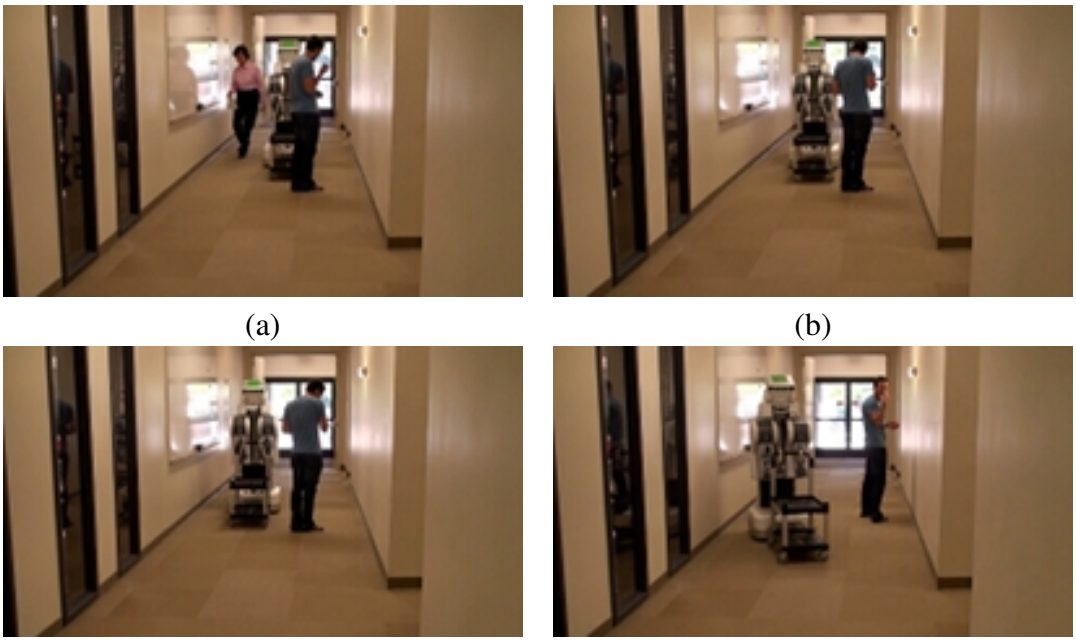
Analysis of the data from the planner showed that it was generally successful in finding paths quickly. The analysis is summarized in Table 4.6 for one run lasting nearly an hour in which we repeatedly directed the robot to goals at far regions of the building. The time to first solution corresponds to the time (in seconds) taken to find a solution with the initial value of $\varepsilon = 3.0$. The number of expands corresponds to the number of states that were



(a) (b)

(c) (d)

Figure 16: Articulating the cart to execute a tight turn.



(a) (b)

(c) (d)

Figure 17: Stopping in presence of person and then replanning to go around him.

expanded in getting to the initial solution ($\varepsilon = 3.0$).

Table 1: Global planner statistics from a long run.

number of planner calls	134
expansions (first solution) (mean)	15157.87
expansions (first solution) (std. dev.)	24529.92
time to first solution (secs) (mean)	1.23
time to first solution (secs) (std. dev.)	1.93
final epsilon (mean)	1.30
final epsilon (std. dev.)	0.61
number of planning failures	25

Of the 134 calls that were made to the planner, 25 did not succeed. An examination of these calls showed that they failed much before the allocated final time for the planner expired. There are several reasons that this could have happened. If the robot was blocked by people, no plan would be found for getting out of a tight spot and the planner would fail. Once the people moved out of the way and the costmap was clear, the planner would be able to find a path to its goal. Spurious sensor readings were another frequent cause of failures. Although we attempted to filter out these readings, they were never completely eliminated and would often completely block all plans for the robot. The recovery behavior helped in clearing out these spurious reading but improving the quality of our sensing is essential to achieve longer robust continuous operation.

The addition of the articulated primitives clearly helped the robot traverse tight corridors, made the plans look much more natural, and reduced the total footprint of the robot as it was making turns. Figure 18 illustrates the plans for a turn through -90 degrees. The motion plan on the left was made with a set of articulated motion primitives while the plan on the right was planned without them.

4.7 Discussion

In this chapter we presented a robust and reliable system for a full-scale mobile manipulation task: cart pushing in a typical office environment. Our method is able to handle

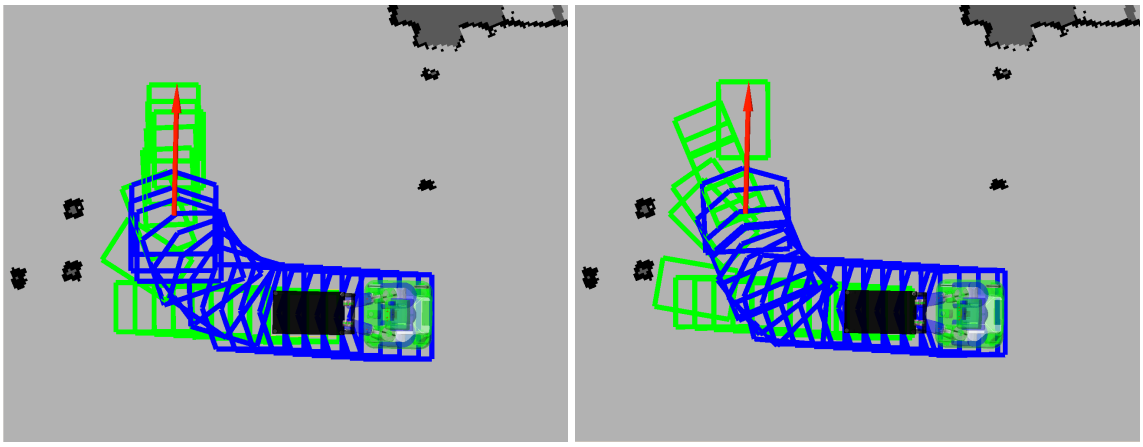


Figure 18: The overall footprint of the robot when using articulated primitives is much smaller (left) than when it is not using them (right). The red arrow indicates the goal for the robot base, the blue polygon is the footprint of the robot during the plan and the green rectangle represents the cart
.

tight turns using articulation of the cart, to escape very tight spots requiring the planning of a series of small moves, and can respond to the presence of dynamic obstacles such as humans.

These capabilities relied on a low-dimensional configuration space that included both the robot and the target object. However, in this configuration space the DOF pose parameters are hard-coded, and the planner assumes that a controller exists which can reliably articulate the target object about this point. This method is therefore not capable of adapting to constraints such as wheels or casters on the objects themselves. For such a capability the robot requires methods for determining, online, the physical model for any object it encounters, as well as suitably general methods for control with respect to these models. In the next chapter we address the model learning problem, and consider the control problem in Chapter 6.

CHAPTER V

LEARNING KINEMATIC AND DYNAMIC CONSTRAINTS

So far we have contributed two insights towards our goal of decision making in unfamiliar physical environments. The first was that in order to adapt to novel objects the agent must be able learn their dynamics online. However, the method we presented operated with a very restricted action space which limited its applicability to more general manipulation tasks. Therefore the second was that it is important to consider the agent's body and articulation abilities to solve more challenging mobile manipulation problems. If the first served to motivate the need for tools from the robotics toolkit, the second gave a taste for how deep this rabbit hole goes. The central problem is the explosion in complexity that arises when engineering systems for full-scale mobile manipulation. This chapter asks whether it is possible to incorporate this domain knowledge in a more parsimonious way.

The goal of this chapter is to provide a principled framework for supporting decision problems in physical domains with under-specified dynamics. Specifically, our aim is to eliminate the implicit keyframe-based constraint representation from Chapter 4 by directly modeling the continuous motion of interacting rigid bodies.

We begin by providing an overview of differential-equation based dynamics, which is the mathematical foundation of *state-space dynamics*, and the main point of departure from motion primitives and other more general transition models in Reinforcement Learning. However, the *state-space* approach only gives a general template for representing systems which can be described by coupled first-order differential equations. For this approach to be applicable to multi-object scenes, the model must capture *relational* behavior – how object movement depends on the state of other objects. We will consider several different approaches to this problem, and ultimately argue for a simulation-based method analogous

to the models of physical reasoning in humans described in Chapter 1.

5.1 State-Space Dynamics

This thesis is concerned with physical planning, such as object manipulation with mobile robots or sprites in physically-realistic video games. The low-level state representation in these domains is typically the position and velocity of one or more rigid bodies. This representation is referred to as the *state-space* representation in control theory, and comes from Newton-Euler dynamics [152]. Actions correspond to the forces and torques that can be applied to these bodies to move them. Standard notation defines a *state-space* in terms of state x and control u:

$$\dot{x} = f(x,u) \tag{15}$$

where $\dot{x}$ is the first time-derivative of the state; however, in order to remain consistent with the RL literature we will use s to denote the state, a to denote actions, and s' to denote next state. This corresponds to the discrete-time version of Eq. 15, as is common in the controls literature, and is obtained by applying $f(s,a)$ for a finite time-interval. When state vectors include more than one object we use superscripts to indicate the state dimensions, (*e.g.* s^i for object i).

In general we assume that the agent can apply forces directly to one object at a time (though objects may interact via contact). Consequently, an action is uniquely defined by a force and torque vector and a target-object identifier. In two dimensions, we require six parameters to represent the state s^i: two for position in the (x,y) plane, one for orientation θ, and three for their derivatives. An action requires five parameters: two for the force $\langle f_x, f_y \rangle$, one for a torque τ_θ, one for a target object index i, and one for a time step δt. For k objects this results in the overall model signature of:

$$f(\mathbb{R}^{6k+5}) \rightarrow \mathbb{R}^{6k} \tag{16}$$

In summary, a state is a concatenation of the position and velocity for each object, and an action is a tuple containing an applied force-torque, target object index, and a time step.

5.2 Object-Oriented Regression

Recall from Chapter 1 that the OO-MDP purports to capture discrete object dynamics in a compact and efficient way. We therefore begin by attempting to integrate *state-space dynamics* into the OO-MDP framework. Our first contribution, Object-Oriented Regression (OOR), combines a non-linear regressor for modeling the continuous dynamics of individual objects with the predicate-based method for representing object relationships from the OO-MDP (effectively creating a hybrid system [53] model). This will constitute a test of the core hypothesis of the OO-MDP that object dynamics can be captured with predicate-based (*i.e.* discrete) description of the possible relationships between objects.

OO-MDP Limitations Recall from Section 2.2.2 that Object-Oriented Markov Decision Processes (OO-MDPs) represent dynamics as a finite set of object attributes and relationships [43]. In this way, OO-MDPs both manage large state-spaces and can generalize to unseen states; however, there are three critical properties that limit the usefulness of OO-MDPs for real-world dynamics:

1. The underlying effect model is discrete, and cannot exploit the geometry of physical state spaces (*e.g.* actions cause *displacements* in *coordinate frames*).

2. It is missing the notion of an *integrator*, which models the evolution of state dimensions conditional on other state dimensions (*e.g.* velocity changes position without external force).

3. Relations are represented with first-order predicates, which cannot define relations parametrically (*e.g.* $contact_\theta(obj_1)$).

As we will show, the first two issues can be overcome by using *state-space regression* as the core dynamics model; however, overcoming the limitations of first-order predicates to represent relationships for real-world applications is a more serious challenge.

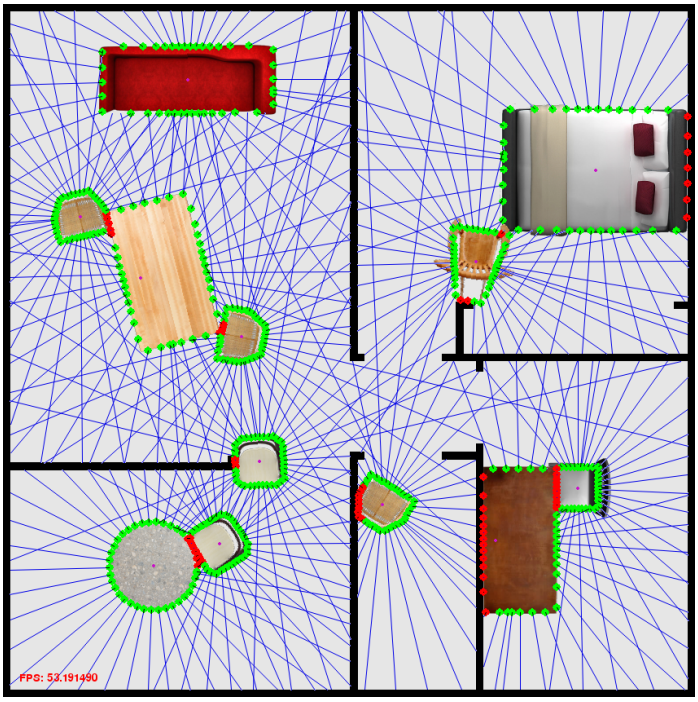


Figure 19: Detecting collision sectors for contact predicates (OO-LWR) in an apartment task.

5.2.1 Collision Predicates

In the original OO-MDP, the choice of contact predicates was driven by adjacency properties of states arranged in a grid; however, in reality objects can come into contact from any orientation, and react according to where the collision occurred in the coordinate-frame of the object. To handle this reality, the object boundary can be discretized into a set of n_s contact sectors, each of which is assigned a contact predicate:

$$contact_{\theta_1}(obj), \ldots, contact_{\theta_n}(obj)$$

Importantly, sector predicates must be computed in the coordinate frame of the object. For example, the reaction of a shopping cart to a collision depends on its direction relative to its wheels, not the grocery store in which it is located.

Fig. 19 illustrates sector-based collision detection applied to objects in an apartment. Lines indicate the level of discretization, and dark (red) dots represent the sectors in collision. The number of sectors n_s is a free parameter of the model.

5.2.2 Local State-Space Models

As mentioned in Section 5.1, *state-space* dynamics are differential, defining displacements from the current state. Objects can also have differential constraints (*e.g.* wheels) causing transitions to be non-linear in the state and action parameters; therefore, the effect model must be (a) compatible with the *state-space* representation, (b) invariant to object pose, and (c) capable of representing non-linear functions.

The building block for *state-space* regression models matching Eq. 16 are scalar-valued predictors of the form $f(\mathbb{R}^{k+5}) \rightarrow \mathbb{R}^1$ for each dimension of each object. To simplify notation here we use the superscript to denote individual state dimensions of a single object, rather than an object index (*e.g.* s_1^1 denotes the x coordinate of object 1 at time $t = 0$, not the full state of object 1). For a single object the function $f(s_t, a_t) \rightarrow s'_t$ can be written as:

$$\begin{bmatrix} f^1(s_t, a_t) + \varepsilon \\ \cdots \\ f^n(s_t, a_t) + \varepsilon \end{bmatrix} = \begin{bmatrix} s_t^{1\prime} \\ \cdots \\ s_t^{n\prime} \end{bmatrix} \tag{17}$$

For these individual predictors we use locally-weighted regression (we summarize the main properties of *LWR* here, but for a more thorough overview see [118]). *LWR* is a kernel-based generalization of linear regression that permits interpolation of arbitrary non-linear functions. In *LWR*, a kernel function is used to compute a positive distance $w_i = k(X^*, X^i)$ between a query point X^* and each element X^i of the training set, which are collected into a diagonal matrix W. Kernels are typically decreasing functions of distance from the query, such as the Gaussian or "squared-exponential": $k(X^*, X^i) \propto e^{-(X^* - X^i)^2/\lambda}$.

Defining the training data $\tilde{X} := [s, a]_{t=0}^T$ and $y := [s']_{t=0}^T$, and the query $\tilde{X}^* := [s^*, a^*]$, state predictions can be estimated for each output dimension with weighted least-squares:

$$\beta_i^* = ((\tilde{X}^T W \tilde{X})^{-1} \tilde{X}^T W y_i' \tag{18}$$

$$s_i' = \tilde{X}^{*T} \beta_i^* \tag{19}$$

In contrast to parametric approaches, the model parameters β^* are re-computed for each

query. As a result, the regression coefficients may vary across the input space, allowing *LWR* to model nonlinear functions with linear machinery.

Pose invariance is achieved by first transforming s'_t and a'_t to the s_t frame, then dropping the position components of s_t. Transforming all observations in this fashion yields a displacement model for individual objects that generalizes across position, at the expense of being able to capture position-dependent effects. However, the only position-dependent effect in the domains we consider is collisions, which are handled by the contact predicates. Furthermore, learning collisions between *dynamic* bodies with *LWR* would require a single monolithic model over the joint state-space of all objects, which would require an infeasible number of observations. Therefore in *OO-LWR* we use a collection of independent single-body, pose-invariant *LWR* models.

In two-dimensions the resulting signature for a single-body *LWR* model is $f(\mathbb{R}^7) \rightarrow \mathbb{R}^6$, which computes a state displacement in the query frame (note that angular velocity is frame-independent in two-dimensions):

$$f(\dot{x}, \dot{y}, \dot{\theta}, f_x, f_y, \tau_\theta, \delta t) \rightarrow (\delta x, \delta y, \delta \theta, \delta \dot{x}, \delta \dot{y}, \dot{\theta}) \tag{20}$$

The overall transition in the original state space is then obtained by transforming the local-frame position, orientation, and linear velocity back to the world-frame. In this fashion, *LWR* can exploit the geometric nature of the *state-space* representation, where coordinate transformations are internal to the regression model (challenge 1). By building a regression model in which a given output dimension can depend on multiple state dimensions, this approach can also effectively handle integration effects (challenge 2). We now discuss how these models can be fit.

5.2.3 Fitting Local Models

Recall that the purpose of predicates in an OO-MDP is to segment the state-action space into sets with distinct object dynamics. The process of training an *OO-LWR* model on a

history of observations $h = [s_t, a_t, s_{t+1}]_{t=0}^{T}$ is therefore achieved by assigning observations to effect models by *condition*, where a condition is a boolean string containing the output of all relations (e.g. collision predicates) applied to that state. For example, all training instances in which the front of a given chair is in collision should be assigned to the same condition. By using *state-space* regression for the individual effect models, *OO-LWR* is able to model rigid body motion for multiple objects; however, *OO-LWR* does not naturally admit a compact representation of the space of possible collisions. As we will see, there are considerable performance implications for larger domains.

5.3 Physics-based Reinforcement Learning

As discussed in Section 5.2.3, Object-Oriented Regression gives means of handling collisions while avoiding any parametric assumptions about the underlying object dynamics. Here we present *PBRL*, a parametric alternative. In *PBRL* the basic idea is to view a *physics engine* as a *hypothesis space* for nonlinear rigid-body dynamics. This representation allows us to compactly describe transition uncertainty in terms of the parameters of the underlying physical model of the objects in the world. We capture this uncertainty using distributions over the relevant physical quantities, such as masses and friction coefficients, and obtain transitions by taking the expectation of the simulator's output over those random variables.

5.3.1 Physical Quantities as Latent Variables

At its core, a physics engine uses systems of differential equations to capture the fundamental relationship between force, velocity, and position. During each time step the engine is responsible for integrating the positions and velocities of each body based on extrinsic forces (*e.g.* provided by a robot), and intrinsic forces (*i.e.* differential constraints).

Differential constraints are ubiquitous in natural environments, and arise whenever bodies experience forces that depend on their configuration relative to one another. A wheel rolling along a surface, a door rotating around a hinge, and a train gliding along a track are

all examples of differential constraints acting on a body. For RL purposes, these parameters provide attractive learning targets that may prove more efficient than more general functional forms, such as non-parametric regression.

In *PBRL* we model the *state-space* dynamics f (Eq. 16) in terms of the agent's beliefs over objects' inertial parameters and the existence and parametrization of physical constraints, such as wheels. Like a standard Bayesian regression model, this model includes uncertainty in the process input parameters (physical parameters) and in output noise. If $f(\cdot;\tilde{\Phi})$ denotes a deterministic physical simulation parameterized by $\tilde{\Phi}$, then the core dynamics function is:

$$s_{t+1} = f(s_t, a_t; \tilde{\Phi}) + \varepsilon \tag{21}$$

where $\tilde{\Phi} = (\tilde{\phi})_{i=1}^{n}$ denotes a full assignment to the relevant physical parameters for all n objects in the scene, and ε is zero-mean Gaussian noise with variance σ^2.

For any domain, $\tilde{\Phi}$ must contain a core set of inertial parameters for each object, as well as zero or more constraints. Inertial parameters define rigid body behavior in the absence of interactions with other objects, and constraints define the space of possible interactions.

In the general case inertia requires 10 parameters; 1 for the object's mass, 3 for the location of the center of mass, and 6 for the inertia matrix; however, if object geometry is known, we can reduce this to a single parameter m by assuming uniform distribution of mass.[1] This is sufficient for our purposes, but for a full parametrization see [6, 119].

We focus on three types of constraints that arise frequently in mobile manipulation applications: anisotropic friction, distance, and non-penetration.

Anisotropic friction is a velocity constraint that allows separate friction coefficients in the x and y directions, typically with one significantly larger than the other. We define an anisotropic friction joint by the 5-vector $J_w = \langle w_x, w_y, w_\theta, \mu_x, \mu_y \rangle$, corresponding to the joint pose in the body frame, and the two orthogonal friction coefficients. Anisotropic

[1] Mass is often parameterized in this fashion in modern simulation tools, such as Box2D [25]

friction constraints can be used to model wheels, tracks, and slides.

A distance joint is a position constraint between two bodies, and can be specified with a 6-vector $J_d = \langle i_a, i_b, a_x, a_y, b_x, b_y \rangle$ which indicates the indices of the two target objects a and b as well as a position offset in each body frame. Distance joints can be used to model orbital motion, such as hinges or pendulums.

Non-penetration, or contact constraints, are responsible for ensuring objects react appropriately when they come into contact. Object penetration is detected during state integration based on object geometry, and is resolved by computing two types of collision-forces. The first is normal to each collision surface, and pushes objects apart. The magnitude of this force is controlled by the coefficient of restitution r, which is a rigid-body property that can be interpreted as "bounciness". The second is tangential to each collision surface, which captures contact friction and allows transfer of angular momentum. This force is proportional to a contact-friction coefficient μ_c.

In general this model-space is over-complete: not all bodies will have both hinges and wheels. The model must therefore allow constraint effects to be added and removed. This can be accomplished by including auxiliary variables for represented components, *e.g.* using a Dirichlet Process prior on constraints; however, this issue can be avoided for cases where the effects of interest can be represented with a finite number of constraints, and where individual constraints can be nullified for certain parameter settings.

We satisfy these conditions by including only a single wheel constraint, and bounding the number of distance constraints by the number of unique pairs of objects. One wheel is sufficient for modeling the bodies typically found indoors, such as shopping carts and wheel chairs, because they have only one constrained axis (multiple coaxial wheels can be expressed by a single constraint). The wheel can be nullified by zeroing the friction coefficients, and the distance constraints can be nullified by setting $i_a = i_b$.

In summary, our dynamics model for a single body is represented by a set ϕ containing the mass m, restitution r, contact-friction μ_c, plus k distance constraints J^d, and a single

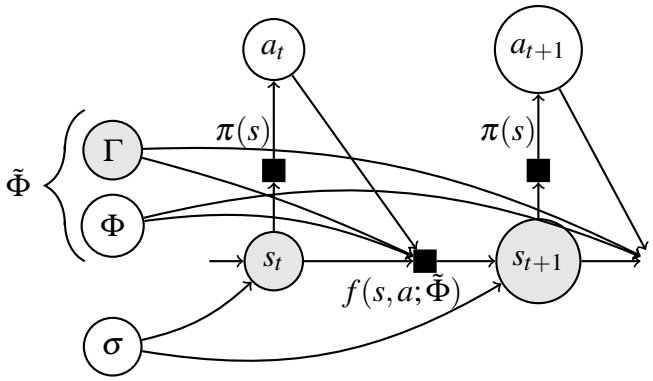


Figure 20: Graphical model depicting the online model learning problem, and the assumptions of PBRL, in terms of states s and actions a. Latent variables Γ (geometric properties) and Φ (dynamics properties) parameterize the full time-series model. $\pi(\cdot)$ denotes the policy and $f(\cdot)$ denotes the dynamics function. We assume Γ to be observable.

anisotropic constraint J^w.

$$\phi := \{m, r, \mu_c\} \cup \{J^d\}^k \cup \{J^w\}^1 \tag{22}$$

Fig. 20 illustrates our approach and modeling assumptions. We split object parameters into two sets according to whether they are potentially observable by the agent. The first, Φ, denotes the *un-observable* physical properties that are needed to parameterize object dynamics, such as friction and mass. The second, Γ, describes geometric information such as polygons or meshes, and are needed to compute inertial forces and collision effects. Note that these both describe physical object *properties*, and are distinct from object *state* parameters (position and velocity). We then define $\tilde{\Phi} = \Phi \cup \Gamma$ as the full set of object properties which are sufficient to parametrize the physical dynamics of all objects in the model.

Inferring Φ from s and a is the model learning problem, and is the focus of this work. Deciding a from s and Φ is the planning problem, which we consider in Chapter 6. Inferring s and Γ from sensor observations is the vision problem, which is outside the scope of this work.

In summary, *PBRL* provides a model prior for object dynamics in terms of a small set of latent physical parameters. The goal of this approach is expressiveness, and the core

Table 2: Univariate distributions for each physical parameter, with $*$ used to indicate subscripting for the appropriate property.

Property $(*)$	**Distribution**
m, r	Log-Normal(μ_*, σ_*^2)
μ_c, μ_x, μ_y	Truncated-Normal$(\mu_*, \sigma_*^2, 0, 1)$
w_x, w_y, a_x, a_y	Truncated-Normal$(\mu_*, \sigma_*^2, a_{min}^{xy}, a_{max}^{xy})$
b_x, b_y	Truncated-Normal$(\mu_*, \sigma_*^2, b_{min}^{xy}, b_{min}^{xy})$
w_θ	Von-Mises$(\mu_{w_\theta}, \kappa_{w_\theta})$
i_a, i_b	Categorical(p_*)

technical challenge is estimating Φ from time-series data, considered next.

5.3.2 A Prior Over Physical models

In order to fully specify a *PBRL* model we must assign priors over each parameter of each body to restrict support to legal values. Mass m and restitution r can take values in $\mathbb{R}^+$, all friction coefficients $\{\mu_c, \mu_x, \mu_y\}$ can take values in $[0, 1]$, all position-offset parameters $\{w_x, w_y, a_x, a_y, b_x, b_y\}$ can take values within the bounds of the appropriate object, orientation $\{w_\theta\}$ can take values in $[-\pi, \pi]$, and index i_a, i_b can take values in $\{1, \ldots, k\}$ for k objects in the world. To represent the agent's beliefs over these parameters, we assign the distributions denoted in Table 2 for each object. In general this model prior would be initialized with uninformative values, and be updated from posterior statistics as the agent receives data.

Fitting physical models Now we consider inferring physical parameters Φ from a history of manipulation data $\{s_t, a_t, s_{t+1}\}_{t=0}^T$. Let h denote a matrix of observed transitions:

$$h = \begin{bmatrix} s_1 & a_1 & s_1' \\ s_2 & a_2 & s_2' \\ \vdots & \vdots & \vdots \\ s_T & a_T & s_T' \end{bmatrix} \tag{23}$$

We should use h to update the the agent's beliefs about Φ and the noise term σ. In a Bayesian approach this is expressed as the model posterior given history h:

$$P(\Phi, \sigma | h) = \frac{P(h|\Phi, \sigma)P(\Phi)P(\sigma)}{\int_{\Phi,\sigma} P(h|\Phi, \sigma)P(\Phi)P(\sigma)} \tag{24}$$

where $\Phi = \{\phi_1, \phi_2, \ldots, \phi_k\}$ is the collection of hidden parameters for the k objects in the domain, and σ is a scalar. This expression is obtained from Bayes' rule, and defines the abstract model inference problem for a *PBRL* agent.

The prior $P(\Phi)$ can be used to encode any prior knowledge about the parameters. In this work $P(\Phi)$ is not assumed to be of any particular parametric form, although this is an interesting avenue for inserting additional structure in physical scenes, *e.g.* that large objects tend to have higher mass, or table-like objects tend to have wheels. For a particular assignment to Φ, Eq. 21 implies a Gaussian likelihood over next states:

$$\begin{aligned} P(h|\Phi, \sigma) &= \prod_{t=1}^{n} P(s'_t | \Phi, \sigma, s_t, a_t) \\ &= \prod_{t=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-\|s'_t - f(s_t, a_t; \tilde{\Phi})\|}{2\sigma^2} \right) \end{aligned} \tag{25}$$

Eq. 25 tells us that the likelihood for proposed model parameters are evaluated on a Gaussian centered on the predicted next state for a generative physics world parameterized by $\tilde{\Phi}$ (*i.e.*, with known geometry and proposed dynamics). Due to Gaussian noise, the log-likelihood for Φ is obtained by summing squared distances between the observed value and the predicted state for each state and action.

$$\ln P(h|\Phi, \sigma) \propto -\sum_{t=1}^{n} \left\| (s'_t - f(s_t, a_t; \tilde{\Phi}) \right\| \tag{26}$$

Note that, technically, any state parameters representing angles can only exist on the interval $[-\pi, \pi]$, and therefore that the likelihood for $(s_t^{2'} - f(s_t^2, a_t^2; \tilde{\Phi}))$ should use a distribution with this support (*e.g.* Von-Mises(μ, κ)). However, in our experiments we ignored this, at the risk of over-penalizing certain predictions with rotation errors that wrap around the unit-circle.

Along with the prior defined in Table 2, this provides the necessary components for a

Metropolis sampler for the unnormalized density:

$$P(\Phi, \sigma | h) \propto P(h|\Phi, \sigma) P(\Phi) P(\sigma) \tag{27}$$

These posterior samples can then be used by any stochastic planner for selecting actions with respect to the agent's model beliefs. Transition samples from a *PBRL* model can be obtained by first sampling the physical parameters Φ from the model posterior, stepping the physics world for the appropriate state and action, and (optionally) sampling the output noise. If $P(\Phi, \sigma | h)$ represents the agent's current model beliefs given a history of observations h, the full generative process for sampling transitions in *PBRL* is:

$$\begin{aligned} \Phi, \sigma \sim & \quad P(\Phi, \sigma | h) \\ \varepsilon \sim & \quad N(0, \sigma^2) \\ s_{t+1} = & \quad f(s_t, a_t; \tilde{\Phi}) + \varepsilon \end{aligned} \tag{28}$$

MCMC for PBRL Despite being compact and self-contained, constructing a sampler for a probabilistic model on a physics engine API is non-trivial. At a minimum it requires specifying prior distributions of the appropriate kind from Table 2, each with with valid support on every object (*e.g.* wheel offsets (w_x, w_y) must fall within the area of the object), and deriving expressions for their conditional likelihood. In our experiments we found it greatly useful to specify the *PBRL* prior using an MCMC library called PyMC [122], which automates the process of constructing Metropolis-Hastings samplers for arbitrary probabilistic models. A full description of the PyMC language is outside the scope of this thesis, but in short it implements a concise syntax for declaring random variables that can depend on other random variables. These variables can be used in subroutines as though they were regular python variables, for example in a deterministic physical simulation that depends on the random variables in Φ. PyMC keeps track of the dependencies of these variables, and automatically computes the overall model log-likelihood. We take advantage of this to construct a scalable model prior that can handle multiple interacting parameters for an arbitrary number of objects.

In our experiments we also discovered a practical issue when scaling MCMC inference to more than one object, resulting in a minor change to the noise term in the model. To understand this, we must first explain the behavior of an MH sampler using a scalar noise parameter $\sigma \in \mathbb{R}$ as defined in Eq. 21. MH defines a random walk in model space, guided by a proposal distribution (*e.g.* adaptive Gaussian) which generates candidate samples for the each model parameter to be accepted or rejected according to the MH ratio [22]. However, when performing MH on regression models like the one here, the noise parameter plays a critical role in determining the likelihood of the data – the larger the value of σ, the smaller the impact of changes in Φ on the overall likelihood[2]. The result is that σ collapses as the model fit increases, decreasing the overall model entropy. As a consequence, the sampler has to pay a greater price for proposals to Φ that are suboptimal, effectively making the sampler greedier over time. This is desirable – indeed a small value of σ indicates that a good model has been found.

The difficulty is that σ only collapses when the **entire** model has been fit. For large models such as the one we consider here, the probability of simultaneously finding good values for all model parameters is very low[3]. Fortunately, we can ameliorate this issue by taking advantage of the object-wise block structure of our model space. Ideally what we want is the model to be able to fit the components of Φ object-wise, in order to match the block structure of the model likelihood.

We can achieve this behavior by replacing the scalar noise parameter with a vector containing one element for each object (*i.e.* $\sigma \in \mathbb{R}^k$). This does not change the validity of original likelihood expression in Eq. 26, but introduces object-wise control of the steepness of this function:

[2]This should be intuitive, since σ controls the width of the Gaussian centered on the model predictions $f(s_t, a_t; \tilde{\Phi})$

[3]Even if the sampler finds a good fit for one object ϕ_i, if the remaining components of Φ are poor then σ remains large and the sampler tends to "walk away" from ϕ_i.

$$\ln P(h|\Phi,\sigma) \propto \frac{-\sum_{t=1}^{n} \left\| (s'_t - f(s_t, a_t; \tilde{\Phi}) \right\|}{\sigma^T \sigma} \tag{29}$$

The results presented in Section 5.5 were obtained without this optimization, but it was necessary for all results in Chapter 6. For similar reasons, we also found it useful to define a proposal distribution for sampling parameters in Φ object-wise (*i.e.* sampling each ϕ_i in turn).

5.4 Model Learning on Real Robots

So far we have considered the problem of modeling and estimating object dynamics in an idealized setting in which the agent has access to the the exact position and velocity of every object in the scene, as well as the force and torque being applied to the object's center-of-mass. In this section we consider the practical issues in obtaining these data from sensors on a real robot, and transforming them to a single consistent reference frame. We also address an unexpected difficulty that arose while attempting to fit models using the likelihood expression in Eq. 26. Further implementation details can be found in [143].

5.4.1 Hardware Prerequisites

There are two basic requirements for implementing our approach on a real robot. The first is a method for measuring the reaction forces and torques during manipulation. The process presented here assumes access to measurements from a wrist-mounted force-torque sensor as well as an end-effector suitable for manipulating the target objects. Although our model as described above was restricted to planar dynamics, we describe a solution involving full 3D sensing and actuation as is typically required in robot manipulation.

We denote the raw 6-axis force-torque measurement in the gripper frame as ${}^{g}_{g}\mathscr{F} = [F, \tau]^T$, which contains linear component $F = [F_x, F_y, F_z]$ and moment $\tau = [\tau_x, \tau_y, \tau_z]$. Following Craig [30], we use the leading subscript to indicate source frame (in which the quantity is located), and the superscript to indicate the target frame (in which the quantity

is observed). For example, we will write ${}^{g}_{g}\mathscr{F}$ to represent the gripper force-torque as measured at the gripper, ${}^{o}_{g}\mathscr{F}$ to represent the same force-torque as seen in the frame of object o, and ${}^{w}_{o}\mathscr{F}$ the same force-torque transformed to the object's center-of-mass and observed in (rotated to) the world frame.

The second requirement is a method for tracking the trajectories of objects over the course of manipulation episodes. In this work we are interested in planar-dynamics, and require the planar pose and velocity of the target object. We denote the 6-dimensional state vector containing object o's position and velocity in the world-frame as ${}^{w}_{o}x = [x_o, y_o, \theta_o, \dot{x}_o, \dot{y}_o, \dot{\theta}_o]$. We use ${}^{w}_{o}T$ to denote the corresponding homogeneous transform, composed of the $3x3$ rotation ${}^{w}_{o}R$ and $3x1$ translation ${}^{w}_{o}P$:

$$ {}^{w}_{o}T = \begin{bmatrix} {}^{w}_{o}R & {}^{w}_{o}P \\ 0 & 1 \end{bmatrix} \tag{30} $$

We also require the world-frame pose of the gripper, ${}^{w}_{g}T$ during the episode for mapping applied forces to the object frame, considered next.

5.4.2 Gathering Data

To gather data the robot must apply manipulation forces to some point on the object, and record the force-torque response as well as resulting object trajectory. To avoid having to introduce the end-effector contact point in the model presented in Eq. 21, the sensor readings must be adjusted to compensate for the weight of the end-effector, and transformed to the object frame.

These are standard operations [30] which we reproduce here for completeness. First, let ${}^{A}_{B}\mathscr{T}$ denote a force-moment transformation which maps force measurements from frame B to frame A:

$$ {}^{A}_{A}\mathscr{F} = {}^{A}_{B}\mathscr{T}\,{}^{B}_{B}\mathscr{F} \tag{31} $$

$$ = \begin{bmatrix} {}^{A}_{B}R & 0 \\ {}^{A}_{B}P \times {}^{A}_{B}R & {}^{A}_{B}R \end{bmatrix} \begin{bmatrix} {}^{B}_{B}F \\ {}^{B}_{B}\tau \end{bmatrix} \tag{32} $$

where ${}^{A}_{B}R$ and ${}^{A}_{B}P$ denote the rotation matrix and translation vector, respectively, from frame B to A, and $P\times$ is the cross-product operator

$$P\times = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix} \tag{33}$$

We can therefore compute the applied force and torque at the center of mass for object i, as seen in the world frame, as follows:

$$\begin{aligned} {}^{w}_{o}\mathscr{F} &= {}^{w}_{o}\mathscr{T}\,{}^{o}_{g}\mathscr{T}\,{}^{g}_{g}\mathscr{F} && (34)\\ &= \begin{bmatrix} {}^{w}_{o}R & 0 \\ 0 & {}^{w}_{o}R \end{bmatrix} \begin{bmatrix} {}^{o}_{g}R & 0 \\ {}^{o}_{g}P\times {}^{o}_{g}R & {}^{o}_{g}R \end{bmatrix} \begin{bmatrix} {}^{g}_{g}F \\ {}^{g}_{g}\tau \end{bmatrix} && (35)\\ &= \begin{bmatrix} {}^{w}_{o}R\,{}^{o}_{g}R & 0 \\ {}^{w}_{o}R\left[{}^{o}_{g}P\times {}^{o}_{g}R\right] & {}^{w}_{o}R\,{}^{o}_{g}R \end{bmatrix} \begin{bmatrix} {}^{g}_{g}F \\ {}^{g}_{g}\tau \end{bmatrix} && (36) \end{aligned}$$

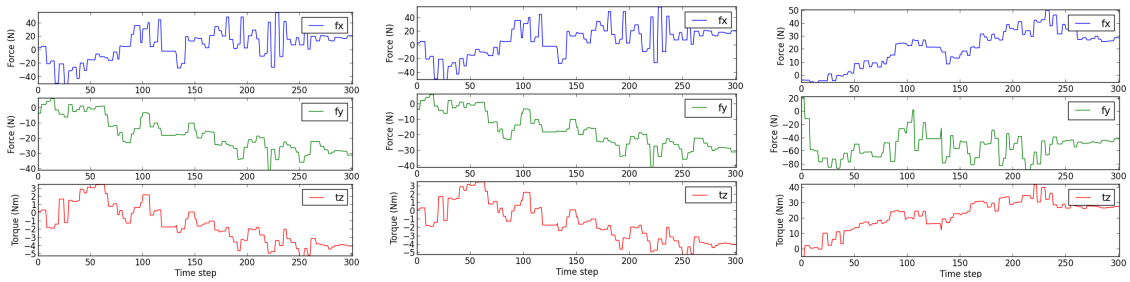
This process can be visualized in Fig. 28, and is equivalent to applying the force-moment transform from Eq. 32 using the transform:

$${}^{w}_{og}T = \begin{bmatrix} {}^{w}_{o}R & 0 \\ 0 & 1 \end{bmatrix} {}^{o}_{g}T \tag{37}$$

Note that the first transform involves a change of location from the gripper to the object center-of-mass, but the latter only involves a rotation. Fig. 21 shows an example of these operations being applied simultaneously to the left and right gripper force-torque signals during the *pull* action on the rectangular table in Fig. 44(g).

Gravity Compensation A straightforward method for gripper mass compensation is to cache an initial sensor offset ${}^{g}_{g}\mathscr{F}_0$ and gripper pose ${}^{w}_{g}T_0$ (before object contact), and transform it to the current frame ${}^{w}_{g}T_t$ using Eq. 32 at each time step:

$${}^{g}_{g}\mathscr{F}^{*}_{t} = {}^{g}_{g}\mathscr{F}_t - {}^{{}^{w}_{g}T_t}_{{}^{w}_{g}T_0}\mathscr{T}\,{}^{g}_{g}\mathscr{F}_0 \tag{38}$$



(a) Left Gripper (b) Right Gripper (c) Object COM

Figure 21: Force-Torque data gathered during pull action on rectangular table with a locked left caster (unknown to robot). (a) Raw readings in left-gripper frame (b) Raw readings in right-gripper frame (c) Total force-torque transformed to the object center-of-mass

This adjusted reading ${}^{g}_{g}\mathscr{F}^{*}_{t}$ can now be transformed to the object frame using the current transform between the gripper and the object, following Eq. 36. This method can handle changes in the pose of the gripper with respect to gravity, which is sufficient for our purposes. We note that it cannot handle changes in conformation (*e.g.* finger movement) or inertial effects which may be relevant for different applications. We also note that in practice the sampling rate of the object-tracker and force-sensor may differ, so Eq. 38 should be computed using the most recent relative transforms from the object tracking system.

5.4.3 Estimation On Real Data

Estimating the *PBRL* model in Section 5.3.2 involved MCMC sampling guided by the log of Eq. 27. The (log) priors $P(\Phi)$ and $P(\sigma)$ were evaluated directly, and typically chosen to be uninformative. The log-likelihood in Eq. 26 was obtained by summing squared distances between the observed value and the predicted state for each state and action. A challenge when attempting to fit $\log(h|\Phi,\sigma)$ in practice is that the trajectory data on real robots will be densely sampled, and contain both noise and un-modeled dynamical effects (*e.g.* static friction or caster orientation). In our initial experiments this lead to large data sets for relatively short manipulation episodes, as well as inaccurate long-term predictions.

To address this, we modify the log-likelihood term to penalize the final integrated error,

rather than incremental displacements:

$$\ln P(h|\Phi,\sigma) \propto -(s_T - s_T^*)^2 \tag{39}$$

where s_i^* is defined recursively as $s_i^* = f(s_t, a_t; \tilde{\Phi})$, with $s_0^* = s_0$.

The main difference with Eq. 26 is that this function maintains a separate predicted state x^* over time. Therefore the actual states x_t in the trajectory are only used to evaluate the transformation of each control input u_i to the object frame (as necessary to omit grasp variables), and final error for $t = T$. This can be understood as a method for handling an under-parameterized model: By not penalizing deviations from intermediate points in the trajectory we allow greater flexibility to fit the overall displacement.

5.5 Evaluation

On this section we evaluate the methods described above both in simulation and on a physical robot using a series of object manipulation tasks. Our goal is to answer the following two questions:

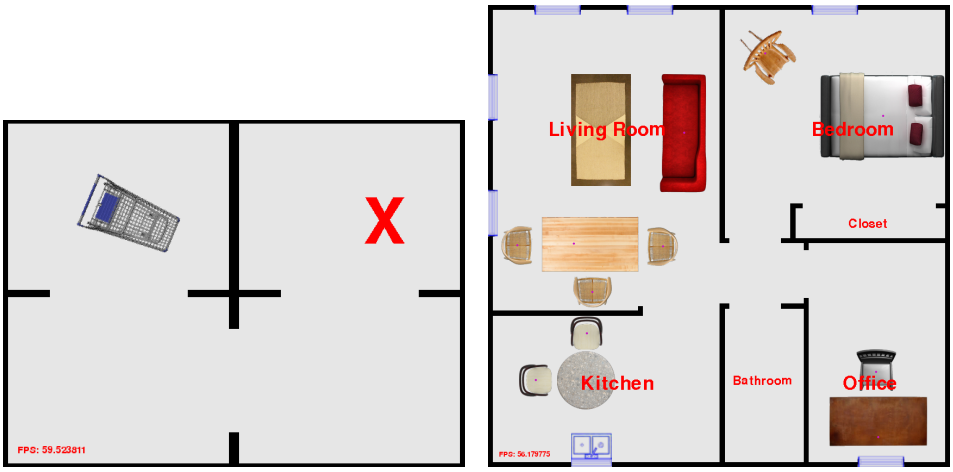
1. What is the most efficient way to capture collision dynamics with multiple constrained rigid bodies?

2. Is the constraint parameterization in *PBRL* is learnable, and if so is it more efficient than non-parametric regression?

These questions will be answered implicitly by examining agent performance, and also by directly comparing model accuracy curves as a function of data quantity and quality.

5.5.1 Simulation Results

Before presenting our results on the robot, we take advantage of the fact that our engine-based approach provides infinite source of synthetic data to rigorously test our proposed methods under a variety of learning conditions. Given our ultimate aim of efficiently solving manipulation tasks in novel environments, our chief interest is in the performance of



(a) Shopping Cart (b) Apartment

Figure 22: Simulated manipulation domains

PBRL and *OO-LWR* on actual tasks. We will also present the results of several experiments comparing these models directly, as a means of explaining the task performance pattern.

Shopping Cart Task The first task is to push a shopping-cart to a user-specified goal configuration. Reward was proportional to the L_2 distance of the cart from the red cross marked in Fig. 22(a). The cart was modeled as a single body with a wheel constraint in the center of the handle axis, and behaved similarly to a real shopping cart which can pivot around points along the wheel axis, but cannot translate along the same axis. Because the cart can collide with the wall, the model must be able to handle collisions. It must also be capable of modeling the non-linear behavior of the cart with sufficient accuracy to produce a plan over the long horizon of the task.

We present results under two learning conditions for this task: one which is noise-free, and a second in which the training observations were corrupted by Gaussian noise ($\sigma = 0.25$). In addition to our primary comparison of *PBRL* and *OO-LWR*, we also include the performance of an agent directly using the *LWR* model described in Section 5.2.2. This was included to decouple the object-oriented approach from the use of an effect model which can model state integration. An agent given access to the true model is provided as

Table 3: Table of the relevant algorithm parameters for each experiment. k: number of nearest neighbors (LWR,OO-LWR), λ: bandwidth (LWR,OO-LWR), n_s: number of sectors (OO-LWR), n_{tps}: number of raycast collision tests per sector (OO-LWR), ε_c: collision radius (OO-LWR), prior: type of prior (PBRL), MCMC: sampler parameters (iterations, burn-in, thin, number of chains) (PBRL).

	k	λ	n_s	n_{tps}	ε_c	prior	MCMC
Shopping Cart	1000	1.5	4	10	20	continuous	2e4,1e3,10,30
Apartment	1000	1.5	4	10	20	categorical	5e3,1e3,10,1

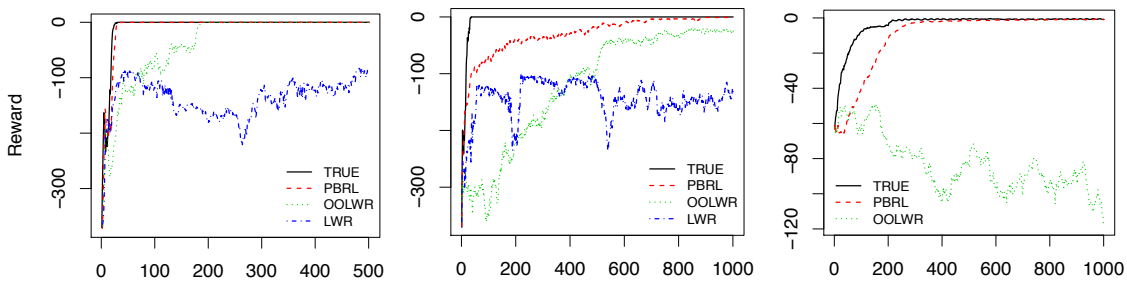
a baseline.

Fig. 23(a) shows the online performance of agents using each of these models in the noise-free condition. Each trace represents an average over 10 episodes. At each step the agent received an observation, updated its model, and selected a new action using an A* planner.

In this experiment we expected the collision-aware agents, *PBRL* and *OO-LWR*, to be successful, and the regular *LWR* agent to get stuck at the closest obstacle *en route* to the goal. Further, we expected the *PBRL* agent to have a steeper learning curve than *OO-LWR*, especially in the presence of noise, owing to its stronger learning bias.

In Fig. 23(a) we observe that in the absence of noise, the *PBRL* agent was able to recover the true model after two steps, and perform nearly as well as the baseline agent. The *OO-LWR* agent was slower to learn, but also reached the goal configuration. We note that despite attaining the same final reward, the online behavior of these agents was very different. Because *OO-LWR* had to (re)learn a separate model for each collision sector, it tended to bump into walls more. This behavior was not penalized in the reward function here, but in situations where collisions are undesirable are during learning, the margin between *PBRL* and *OO-LWR* could be considerably higher.

The *LWR* agent failed to reach the goal configuration because it lacked the ability to model collisions, due to the lack of collision predicates. It was therefore greedy with respect to the reward function, and the value at which it plateaus corresponds to the distance of the wall separating the start and goal configurations. Note that a non-pose-invariant *LWR*



(a) Shopping-Cart task (b) Shopping-Cart with noisy training data (c) Apartment task

Figure 23: Online performance of various agents under different domain sizes and training conditions.

model is capable, in principle, of learning collision dynamics. However, it is incapable of generalizing to unseen states and therefore cannot return useful predictions for states in which the agent has not already visited.

In the presence of training noise (Fig. 23(b)) we observed the same overall pattern of results, but with more gradual learning as was required to average out the noise, in particular for *OO-LWR*. What appears to be a small steady-state error for the *OO-LWR* agent was in fact due to this trace averaging across runs, some of which had not obtained sufficient accuracy to plan a successful path around the wall. The disproportionate effect of noise on *OO-LWR* will be the topic of Section 5.5.1.

Apartment Rearrangement Task The next task is a multi-object rearrangement problem in a simulated apartment. Multi-object rearrangement tasks of this sort are the main focus of this thesis, and provide a true test of the collision-handling capabilities of both methods. The apartment task contains 11 objects with various shapes and physical properties, including fixed wheels (dining table, office desk), large mass (couch, bed), small mass (chairs), and a revolute constraint (kitchen table). Reward was again proportional to the L_2 distance of the objects from user-defined goal configurations, visualized in Fig. 22(b).

The prior for *PBRL* is a categorical distribution defined over a collection of pre-learned modes from individual object trials. This was done because sampling a joint set of object

parameters under the continuous prior in Table 2 using MCMC was very slow to mix. Addressing this issue with more sophisticated mixture-based priors and sampling methods will be a topic for future work.

Fig. 23(c) compares the online performance of *PBRL* and *OO-LWR* on the apartment task. In this domain, the inefficiency of *OO-LWR* is apparent: even after 1000 observations the *OO-LWR* agent was incapable of modeling domain dynamics with sufficient accuracy to produce a valid plan. This result is not surprising, given that *OO-LWR* requires $2^{|O|n_s}$ separate effect models to fully describe the collision space over $|O|$ objects. However, the physics-based representation of collision dynamics yields qualitatively different behavior. In contrast to the predicate-based approach, the *PBRL* agent quickly obtained an accurate estimate of full relational dynamics of the task, and produced a viable plan.

Model Quality In this section we take a closer look at the learning behavior of the model in PBRL, in contrast to regression based alternatives. This section serves to asses the learnability and efficiency of the constraint parameterization in Section 5.3.2. We compared *PBRL* to Locally-Weighted Regression (LWR) [7] and Linear Regression (LR). These methods were selected because they represent common approaches at two different extrema of the bias-variance spectrum, and have complementary strengths:

- *LWR* is a non-parametric method with high expressive power, but requires many data points.
- *LR* is a parametric method which can only represent linear functions, but is very sample efficient.

Both regression methods were trained on data from Eq. 23. In *LR* the free parameters were the regression coefficients, and for *LWR* the free parameters were the Gaussian kernel hyper-parameters.

Fig. 24 compares the efficiency of each model across the ten objects depicted in Fig. 25. The five left-most objects contained a nonlinear constraint in some configuration. The

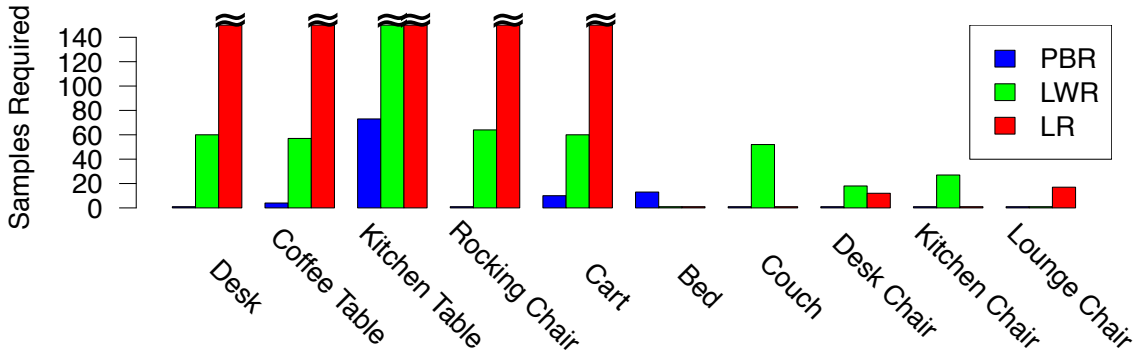


Figure 24: Number of samples required to achieve $R^2 \geq 0.995$ on a collection of household objects

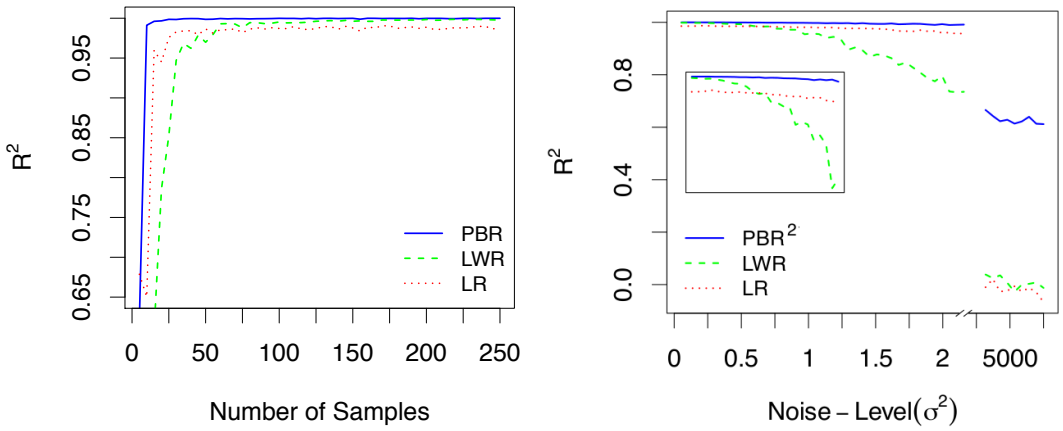
Figure 25: Visualization of the 10 objects used for the evaluation.

remaining objects were linear and only vary in shape and mass. For each object, 50 noisy training observations were gathered online and used to train a model ($\sigma = 0.50$).

For each object, bar height indicates the number of observations required by the model to obtain a given test accuracy, as measured by an r^2-statistic evaluated on 200 test samples. The samples were generated by the same procedure as used for training. The accuracy threshold was set to $r^2 \geq 0.995$. Although it may appear strict, this threshold was empirically determined such that a planner was able to solve a short (50 step) open-loop pushing task around a static obstacle with final error less than the object radius. This criterion is appropriate for the intended purpose of our model, which is to determine kinematically feasible trajectories for manipulation with velocity or force controlled effectors.

As expected, *LR* fails to achieve the target accuracy for all five non-linear objects. However, it is efficient for the remaining ones. *LWR* achieves the accuracy threshold for all objects except the kitchen table. *PBR* was the most sample-efficient model, and reached the accuracy threshold for all objects. However, it was the worst-performing model on the bed object. Since the bed model was representable, this indicates a failure in the MCMC estimator. We plan on investigating the use of different estimators in future work.

Signal-To-Noise Ratio An important practical consideration when fitting dynamics models from data is robustness to noise. As mentioned in Section 5.5.1, we often observed that despite noisy data, *PBRL* agents would find approximately-correct models with a much small number of samples than *LWR*-based agents. In order to measure this effect more closely we examined model test-accuracy as a function of both training noise and training size on data gathered from a simulated shopping cart. Each plot compares three models at different point in the bias-variance spectrum, as discussed in Section 5.5.1. The first, *PBR*, uses the physics-based model defined in Eq. 21. The next two are state-space regression methods using either linear regression (*LR*) or locally-weighted regression (*LWR*) as the underlying regressor.



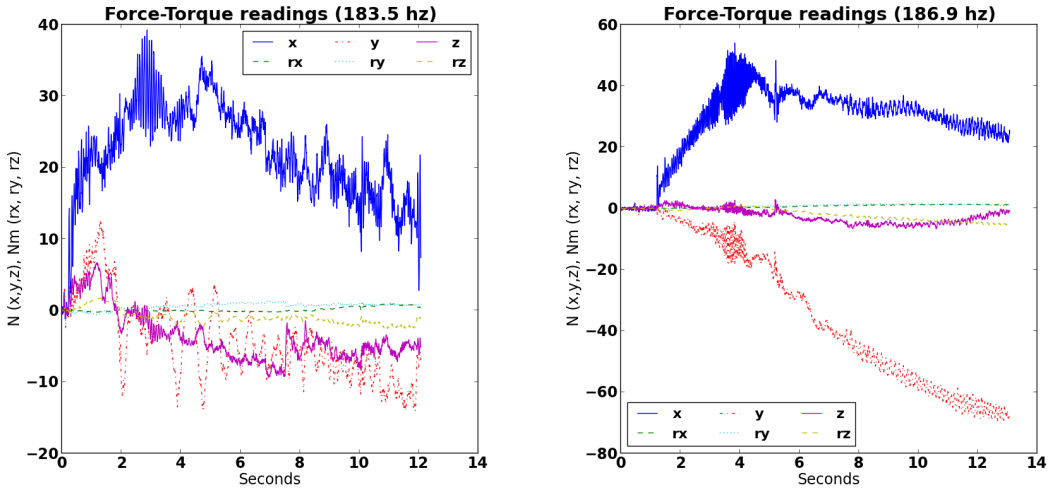
(a) Training Noise (b) Training Size

Figure 26: (a) Model accuracy as a function of training size ($\sigma^2 = 0.50$) (b) Model accuracy as a function of noise level (50 training samples)

Fig. 26(a) shows a a plot of model test-accuracy vs. training size for a fixed noise level (0.50). Fig. 26(b) shows complimentary result, plotting test-accuracy vs. training noise, with number of training samples held constant (50). In both cases the R^2 statistic was computed on 200 test samples independently generated from the simulator with no noise.

The first result in Fig. 26(b) demonstrates that *PBRL* is more robust to Gaussian noise than both regression methods (in cases where it can represent the target dynamics). The inset shows that *LR* fails to reach zero-error even in the absence of noise, and indicates that while *LWR* can achieve zero-error with no noise, its performance quickly falls off as noise increases.

The second result indicates that the variance-related error in MCMC model estimates drops rapidly with subsequent observations. Least-squares for linear-regression (LR) is also a low-variance estimator, as shown by the sharp performance curve. The plateau at a non-zero error is an indication of the model bias in *LR*, which cannot represent the non-linear effect of the wheel on the shopping cart. By contrast, *LWR* can represent the target function, as evidenced by its convergence to $R^2 = 1$, but its predictions suffer more from noise corruption at small training sizes than the other methods. This could be remediated by tuning the kernel bandwidth to average across a wider region, but this would introduce



(a) Table with locked caster

(b) Utility Cart

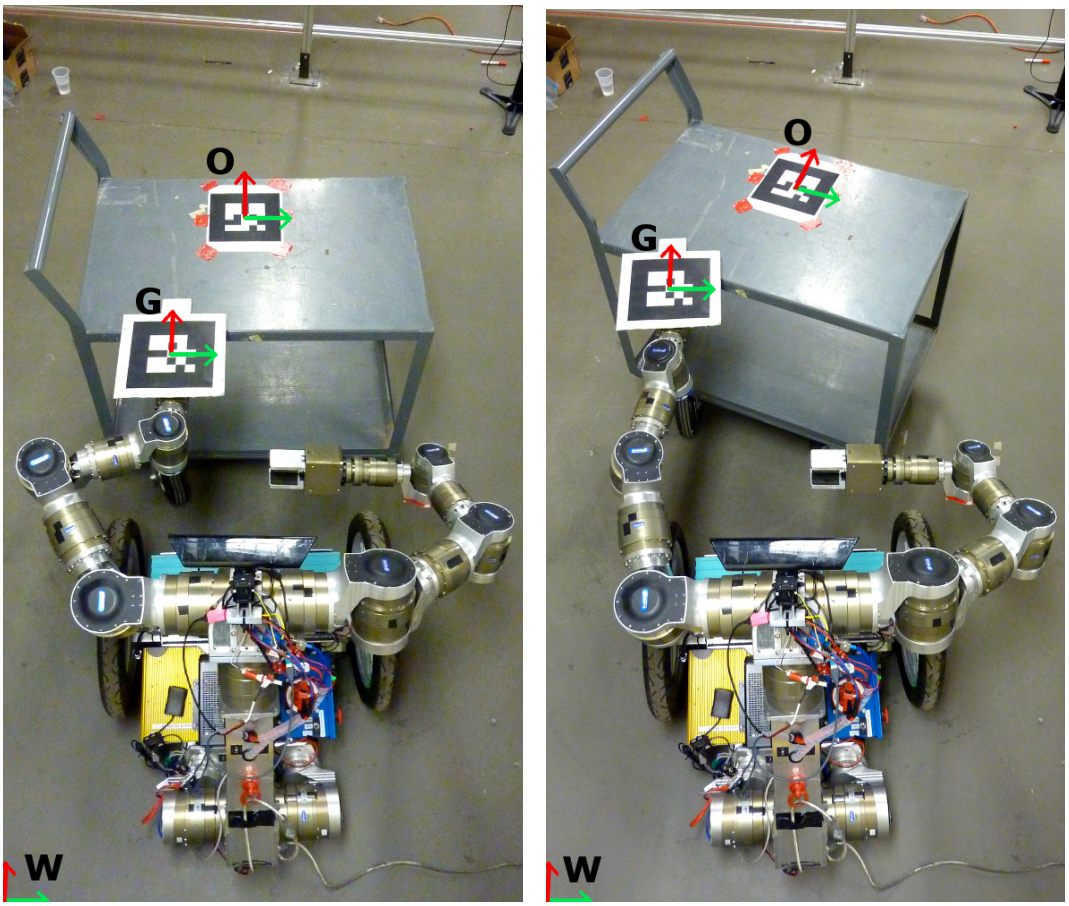
Figure 27: Raw force-torque readings from robot effector during two manipulation episodes. Robot executes a forward (+X) velocity action to (a) a table with a locked wheel, and (b) a utility cart.

a linear bias that shifts the performance of *LWR* closer to *LR*. Overall these results suggest that if an appropriate physics-based model can be defined, it can outperform more general-purpose alternatives.

5.5.2 Robot Results

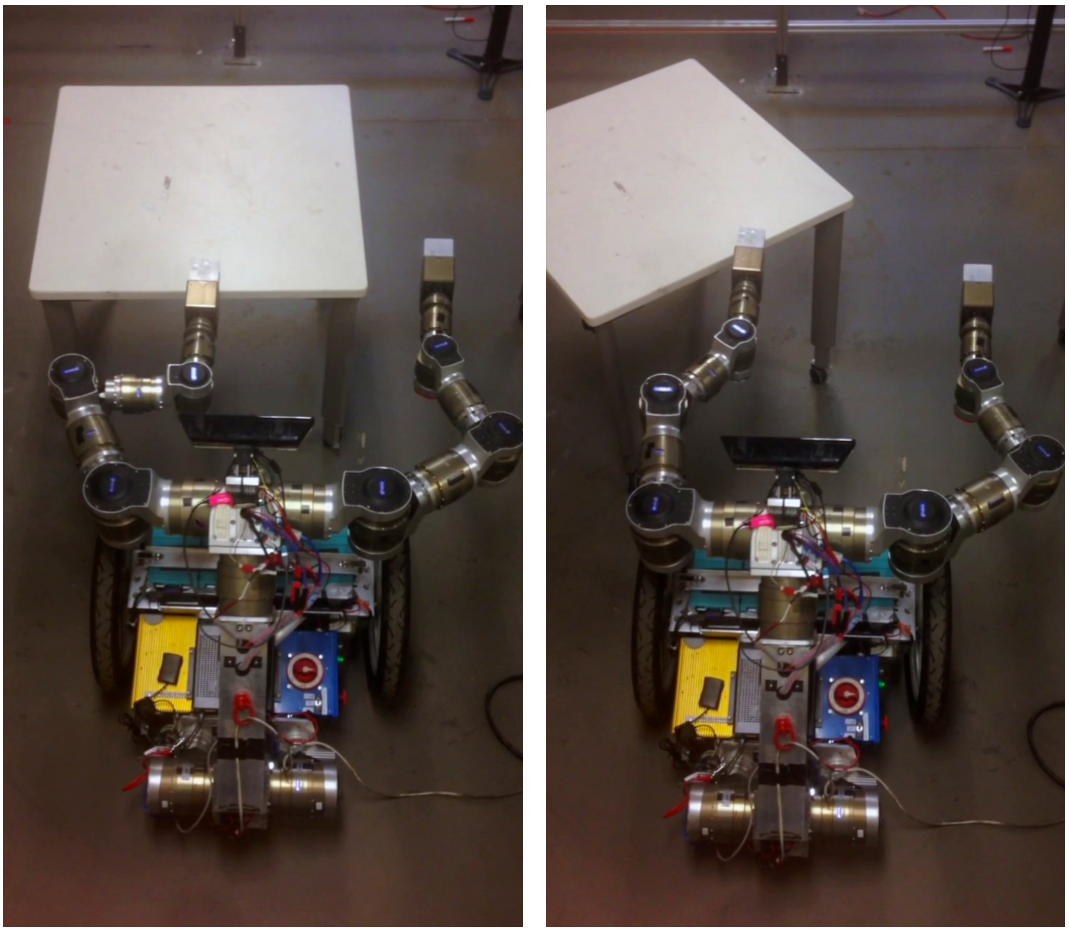
We implemented the proposed framework on the mobile manipulator Golem Krang [164]. As will be covered in greater detail in Section 6.1, Golem Krang has a segway-like base with a slider in the back to provide static stability, two 7-DOF arms with 6-axis force-torque sensors in the wrist joints and 1-DOF gripper end-effectors. To obtain position estimates in the world coordinate system, we used external sensing consisting of 4 cameras for tracking AR-Markers on the robot end-effector and environment objects.

As discussed in Section 5.4.2, the main difference between these experiments and those in [144] is that here we consider actions as measured by a robot-mounted force-sensor that is potentially in non-rigid contact with the target object, rather than simulated forces defined at the object center-of-mass.



(a) Start State (b) Result

Figure 28: Humanoid robot manipulating a cart with a non-holonomic constraint. (a) Start state; robot applies force in the forward (+X) direction. (b) Result: cart rotates and slides along wheel direction.

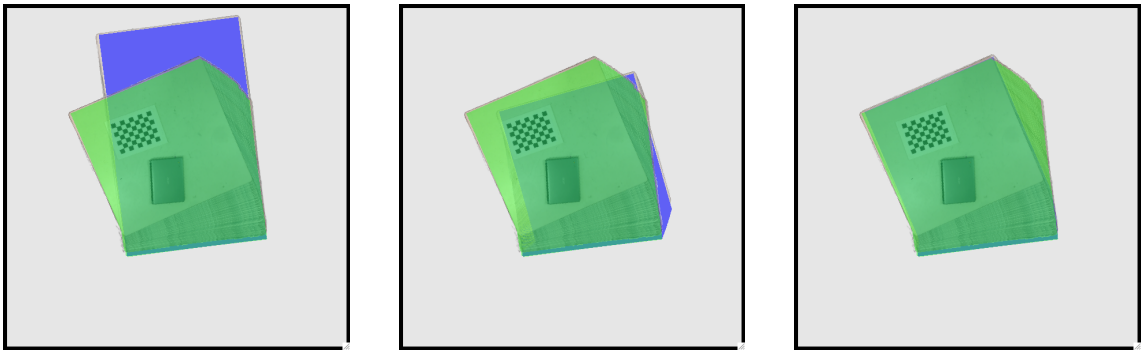


(a) Start State (b) Result

Figure 29: Mobile manipulator pushing an office table with lockable wheels. (a) Start state; robot applies force in the forward (+X) direction. (b) Result: table rotates around locked wheel.

We performed a series of experiments using different configurations of a standard office table, and on a utility cart with fixed front wheels. The robot was tasked with applying a closed-loop velocity controlled push-action on the objects, and estimating the physical parameters of the object based on the force-torque readings as well as position estimates.

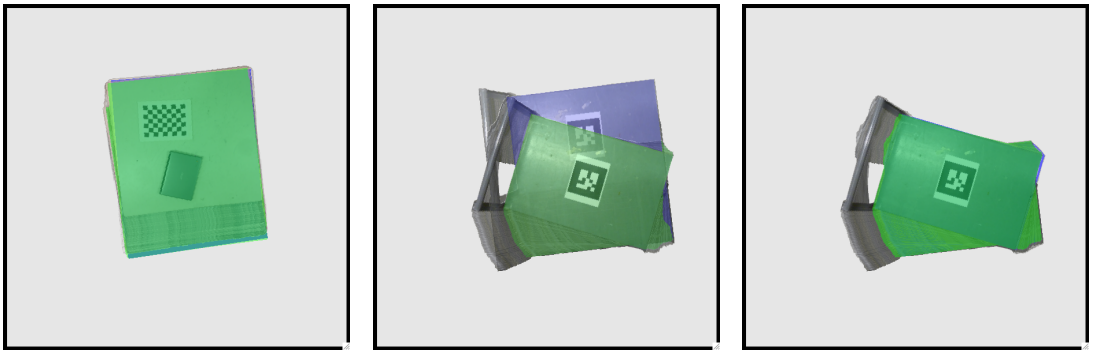
Fig. 29(a) and Fig. 29(b) show the starting and final configurations for an experiment on an office table with lockable wheels. As the robot did not know *a priori* that the lower-left table wheel was locked, the expected outcome was that it would move in a straight line. Fig. 30(a) visualizes the expected and actual obtained behavior.



(a) Expected result: table translates

(b) Attempted learning: inappropriate loss function

(c) Correctly fitting table with locked wheel



(d) Fitting unlocked table

(e) Expected result: cart translates

(f) Fitting a cart with fixed wheels

Figure 30: Overlay of observed data (green) and model predictions (blue) with and without PBR model learning. (a,e) Without learning: robot expects object to move straight forward. (b) With learning, using the full trajectory likelihood function: large error in final position estimate. (c) Learning on locked table: robot estimates that a wheel-constraint is active on the lower-left corner. (d) Learning on unlocked table: robot correctly estimates a mass and friction that reproduce the observed trajectory. (f) Learning on utility cart: robot estimates a fixed wheel on the right side of the cart.

The presented framework uses the observed behavior as well as the obtained force-torque readings, shown in Fig. 27(a), to estimate a model. The first attempt at model learning employed the loss function utilized in Eq. 26. As can be seen in Fig. 30(b), this approach failed to produce accurate long-term predictions when applying the measured force-torque signal to the starting table configuration.

Utilizing the integrated-error loss function Eq. 39, our method obtained a model for the object that included a wheel in the lower left quadrant of the table, with high friction in the push direction. This result consistent with the training data, but and yielded a constraint pose near the location of the actual locked wheel on the table. Fig. 30(c), visualizes an overlay of the observed table positions and a physical simulation of the estimated model when the same forces are exerted on it. In a second experiment in which all wheels were unlocked we obtained the results in Fig. 30(d), in which the mass and friction were tuned to produce the appropriate straight-line trajectory.

Fig. 30(e) shows a similar false prediction on a utility cart, resulting in the updated model shown in Fig. 30(f), containing a fixed wheel.

5.6 Discussion

The goal for this chapter was to define a compact and learnable parameterization of multi-body dynamics for manipulation applications. We presented two methods, *OO-LWR* and *PBRL*, and conducted experiments to determine which approach was more efficient at handling collision effects and nonlinear constraint effects. By contrast to the methods presented in Chapter 3 and Chapter 4, both of these methods were directly based on the differential equations relating position, velocity, and force.

Our results showed that, for small problems, *OO-LWR* could learn collision dynamics and non-linear effects. The primary drawback of *OO-LWR* was that it required large amounts of data to make accurate predictions. Importantly, *OO-LWR* also inherits the general limitation from the use of non-parametric regression that the computation time for

model queries scales (at least) linearly with the amount of training data. Although we didn't consider this issue in this chapter, it is of practical concern for robots operating on real-time budgets.

PBRL pushes the physics-based approach further, adding explicit differential constraints on object motion, and utilizing a proper Newton-Euler integrator for solving them. Compared to *OO-LWR*, *PBRL* encodes much richer domain knowledge about collision effects. Instead of having to relearn object dynamics from scratch for every possible collision, it only has to learn two coefficients: restitution and contact friction. This allowed better generalization, sample efficiency, and query speed than *OO-LWR*.

Our experiments confirmed that the physics-based model in *PBRL* is learnable from data available during robot manipulation. We also showed that *PBRL* is more robust to training noise than *OO-LWR*, and is significantly more sample efficient than non-parametric regression for learning individual object dynamics, leading to improved online task performance. Another potential advantage of *PBRL* is that it contains explicit constraint parameters, rather than relying on the data to represent non-linear behavior. As we will show in Chapter 6, this can be useful when implementing controllers to manipulate these objects in kinematically feasible ways.

CHAPTER VI

MOBILE MANIPULATION WITH LEARNED PHYSICAL CONSTRAINTS

In Chapter 4 we presented a framework for articulated mobile manipulation planning with a two-armed mobile robot, but left open the problems of how to *identify* and how to *plan* with constrained objects when necessary. Chapter 5 addressed the first issue of identifying certain kinds of non-holonomic behavior in indoor scenarios. In this chapter we address the second issue of how to plan with objects from the model class in Chapter 5. We present a method in the spirit of Chapter 3, in which the agent plans with respect to its model beliefs, and updates its model online as it gathers experience in the world.

This chapter also details the implementation of a full mobile manipulation system which applies the methods in this thesis to the Navigation Among Movable Objects problem. In the process, we develop the planning and control machinery necessary to manipulate objects in the *PBRL* model class using a real robot. We provide a complete overview of the NAMO problem and the control stack on our robot, *Golem-Krang* Fig. 31.

6.1 Platform

Krang is a humanoid robot with a 16 degree-of-freedom body consisting of two 7DOF arms, a waist joint, and a torso joint Fig. 31. It has differential-drive base, on which it can balance (Fig. 31) or sit (Fig. 44(a)). In this work we used 1-DOF grippers with large rubber plates for high-friction grasp. The arms are equipped with six-axis ATI force-torque sensors, which are capable of better than $\frac{1}{8}$ N force resolution and $\frac{1}{300}$ Nm torque resolution.

At the lowest level we control Krang's wheel motors and body modules using current or velocity commands. From there we have implemented a base controller that tracks 2D pose

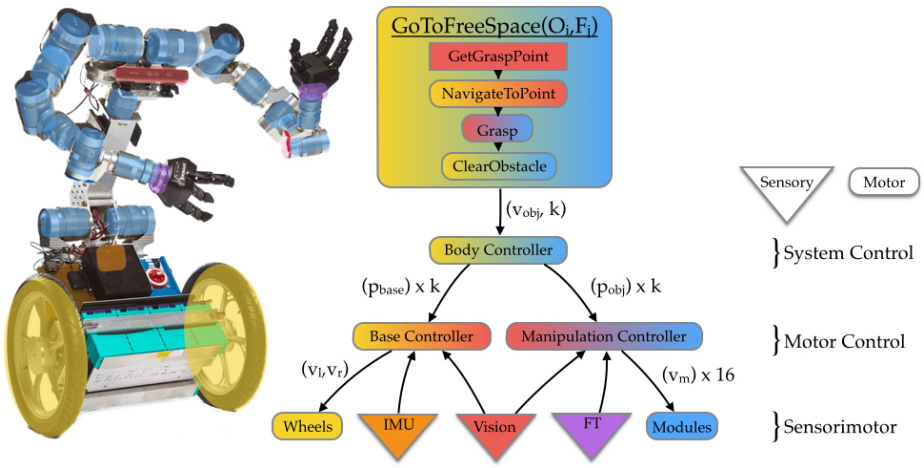


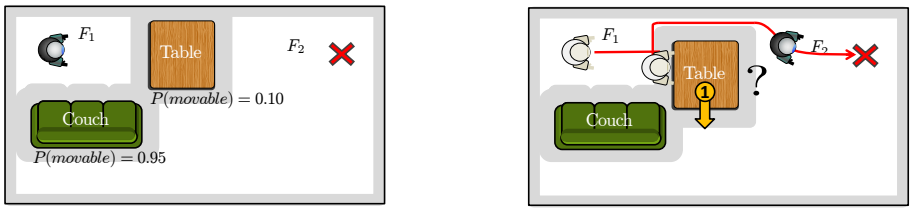
Figure 31: Golem-Krang robot and control diagram. Krang has a 16-DOF body with a differential-drive base, and is actuated by a control stack that achieves whole-body motion using visual servoing with force-feedback.

on the ground using odometry and/or visual localization feedback. For object manipulation we have implemented Jacobian-based Cartesian gripper controllers for both arms, which accept 3D pose and velocity commands in the robot-base frame, and dispatch low-level velocity commands to the modules. This controller is also capable of force-compliance, achieved using FT values from the ATI sensor to drive a proportional offset from the current gripper reference.

6.2 Navigation Among Movable Objects

There is great interest in robots that can safely navigate in common environments such as homes and offices. However, the presence of obstacles poses a serious challenge. As evinced by Chapter 4, interacting with a single piece of clutter found in typical environments is difficult in itself, and the robot may need to manipulate many pieces of clutter to clear a path to a goal safely. Even given a map of the room, how does the robot decide which path to take, or which object to move? This problem is referred to as Navigation Among Movable Obstacles (NAMO) [161, 163, 177]. NAMO is an important research area that is on the critical path to robot interaction in human environments.

There are two primary challenges in developing a practical algorithm for the NAMO



(a) Configuration

(b) Solution obtained through a deterministic NAMO planner.

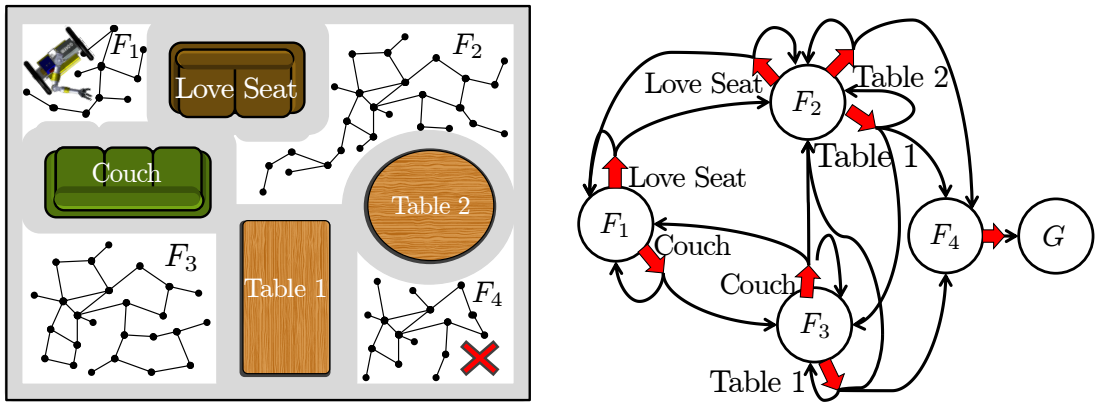
Figure 32: The table wheels are likely to be locked, making it impossible for the robot to move the table. In contrast to deterministic planners, our proposed framework accounts for this probability.

domain: the exponential size of the search space and the inevitable inaccuracies in perception as well as actuation that occur on physical systems.

In a parallel line of research we have presented the first stochastic planning method for solving NAMO problems in realistic environments with model uncertainty [94, 95]. To better understand the importance of handling uncertainty, consider the example in Fig. 32. Perhaps the robot knows that the shortest path to the goal involves moving the table, but it cannot see 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 manipulating the couch? These sorts of reasoning patterns are not expressible within the framework of deterministic search, without resorting to *ad hoc* heuristics.

Leveraging ideas from decision theory, this work has achieved a novel representation that formally addresses uncertainty in NAMO, while simultaneously addressing the dimensionality problem by exploiting task structure in the form of a hierarchical MDP. By casting the NAMO problem as a hierarchical Markov Decision Process, we achieved the first NAMO planner which can bias its decisions at *plan time* in order to compute policies that are *likely to succeed*.

The core technical idea behind this method is the NAMO-MDP, a hierarchical MDP model for capturing the abstract subproblem of moving between *free space regions*. This



(a) Robot determines free space regions as subgraphs in a PRM.

(b) Resulting MDP.

Figure 33: Robot determines free space regions as subgraphs in a PRM and constructs MDP accordingly.

construction allows the planner to focus effort on the relevant part of the state action space. In this section we provide an overview of the NAMO-MDP, summarize the key theoretical and empirical results from [94] and [95], and present an RL-based method for solving it in physical environments with model uncertainty, using ideas from Chapter 4 and Chapter 5.

6.2.1 The NAMO MDP

The NAMO MDP is an "approximate" MDP that closely resembles the real problem but can be solved in linear time for typical environments [94]. 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. This suggests also 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. This property is the basis for the state and action spaces in the NAMO MDP, visualized in Fig. 33.

The second insight is that we can capture the transition and reward dynamics in this abstract representation by computing a low-level manipulation policy for each abstract action. The transition model, $T^a_{ss'}$, is used to model uncertainty in *manipulation dynamics*, such as might occur if obstacle mass or friction is unknown, and should reflect the likelihood of actually creating an opening between the free spaces. The reward function, R^a_s, reflects the expected reward (or cost) in terms of required time and physical work for creating such an opening.

Together these two ideas permit the construction of a hierarchical MDP, which can be solved in a manner analogous to MAX-Q [40]. The obtained policy describes both a mapping for the abstract representation – from a given free space region to an object to manipulate, as well as the raw representation – from a given obstacle state to a parameterized controller.

NAMO MDP Construction The proposed NAMO MDP has a two-level hierarchy, with navigation between regions as the high-level task, and object manipulation as the low-level task. Here we define both MDPs, and their hierarchical semantics. Recall that an MDP is defined as $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$, which leaves four properties to define at each level of the hierarchy (γ can be viewed as a parameter of the algorithm).

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 low-level MDP M_{ll} in terms of these states S_{ll}, and a set of primitive actions A_{ll} for manipulating obstacles on the map. Note that defining a low-level MDP does not imply we are actually solving for optimal low-level policies. Following [94] and [95], we use approximate Monte-Carlo based planners to solve the manipulation problems, and use these estimates in a value-iteration algorithm for the high-level MDP.

For the high-level MDP M_{hl}, we define the state space S_{hl} to be the set of free-space

regions implied by the starting obstacle configuration. To construct S_{hl} we adopt a sampling approach that has been proven in the planning literature and build a roadmap over the state space. The main insight behind this approach is that the resulting roadmap will contain disconnected subgraphs for precisely the free space regions that we are attempting to identify. The probabilistic roadmap (PRM) [75] algorithm samples random configurations within the state-space and connects nearby collision-free samples if there is a collision-free path between them. Constructing a PRM in a disconnected configuration space will consequently return multiple disconnected subgraphs. Each of these subgraphs now encode separate free space regions and together they yield the MDP states. This is visualized in Fig. 33.

Having determined the MDP states, the actions need to be determined. The action space A_{hl} is the union of all possible manipulation sub-tasks for each region, where subtask $a_{hl}^{ijk} = \pi_{ijk}$ means "open a path from state i to state j by manipulating obstacle k". This is done by finding the obstacles disconnecting two 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 subgraphs, the obstacle is considered disconnecting the free spaces and an according action in the MDP is created.

Transitions and Rewards: The rewards in M_{ll} and M_{hl} are defined to reflect their hierarchical relationship. Values for R_{hl} represent expectations over the reward accumulated by subtasks in A_{hl}, and individual subtasks a_i receive the value of their final state in M_{hl} as a terminal reward [94]. This should be intuitive, since the high-level policy needs to know the actual outcome of executing each possible subtask, and each subtask needs to know its *context* in the high-level policy in order to know how important different obstacles are. Initially, only the state $s \in S_{hl}$ containing the goal configuration has a reward, set according to the robot's utility function.

The transition probabilities in M_{hl} directly represent the expected outcome of executing a subtask π_{ijk} in some state s^i_{hl}. For example, the manipulation sub-task π_{ijk} terminates in state j if it successfully opens a path from i to j, and terminates in state i otherwise. Therefore, the transition probability $P(s' = j|s = i, \pi_{ijk})$ is nonzero if and only if π_{ijk} can terminate in j for some *PBRL* model beliefs. In addition, this suggests that the transition model T_{hl} is sparse: the probabilities are automatically zero for all states that do not share an obstacle in their adjacency lists. Transition probabilities T_{hl} and Rewards R_{hl} are estimated by Monte-Carlo simulation of obstacle-clearing plans under sampled model beliefs, following Eq. 28:

1. Sample world w with object parameters $\Phi, \sigma \sim P(\Phi, \sigma|h)$ 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.

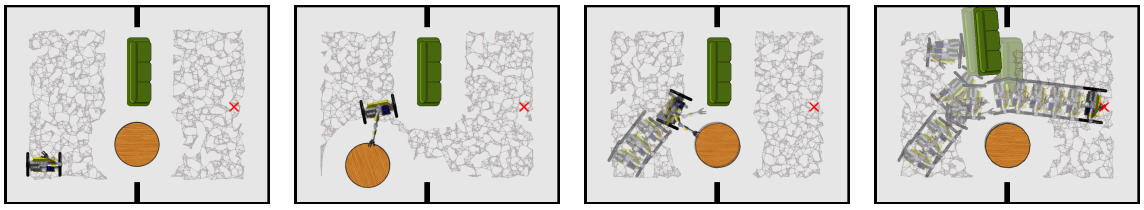
$T^a_{ss'}$ is now set to be the ratio of manipulation plans that succeeded in creating an opening.

$$T^a_{ss'} = P(s' = target|s, a) = \frac{|succ|}{k} \tag{40}$$

By construction, the reward function R^a_s includes the cost for the manipulation action plus a discounted reward for the region reached. Therefore we want the manipulation cost to be negative, and on the same order of magnitude as the terminal reward. We achieve this with a decreasing sigmoid function that is scaled to the range $y \in [0, \alpha]$ for total time t_{total}, total linear force f_{total}, and total torque τ_{total} applied during a manipulation action:

$$R^a_s = \frac{1}{3}\sum_{i=1}^{3} \frac{\alpha \exp(\beta(2\frac{x_i}{m_i} - 1)/2)}{1 + \alpha \exp(\beta(2\frac{x_i}{m_i} - 1)/2} + V_{s'} \tag{41}$$

for quantities $x = [t_{total}, f_{total}, \tau_{total}]$ with nominal maxima $m = [t_{max}, f_{max}, \tau_{max}]$. The shape parameters α and β control the magnitude and steepness, respectively, of the cost function. In the experiments in Section 6.5 we set $\alpha = \frac{200}{4} = 25$ (for terminal reward 200) and $\beta = 10$.



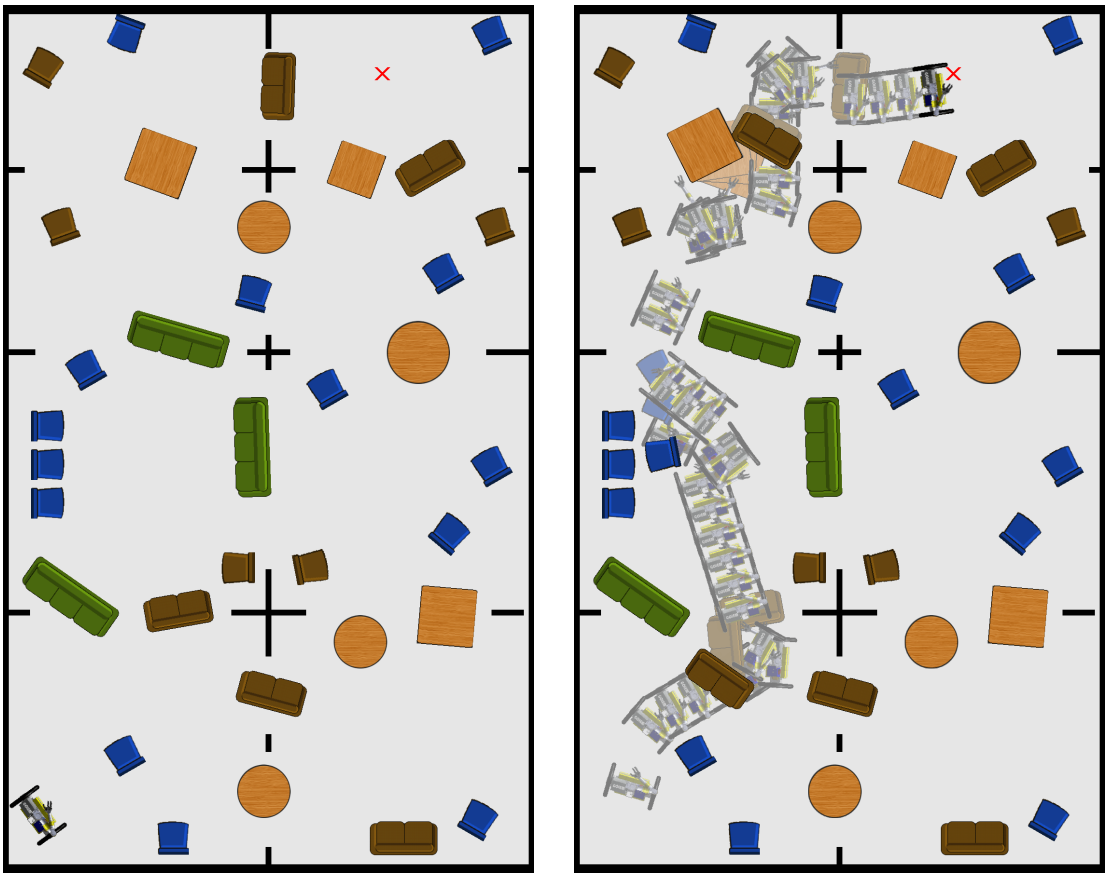
(a) (b) (c) (d)

Figure 34: Adaptive behavior in simulated NAMO task (a) Initial configuration (b) Expected outcome: table has lower mass and offers lowest-cost manipulation plan. (c) Actual behavior: table rotates in place. Robot updates beliefs to reflect high probability of revolute constraint. (d) Based on the new information the robot decides to move the couch.

Note that we cannot just fix a manipulation plan and evaluate it for different samples of $P(\Omega)$ as this would yield an estimate of the success probability of a specific plan rather than provide insight into the general success probability of manipulating the object. Given these estimates, M_{hl} can be solved using standard value iteration.

This completes the construction of the NAMO MDP. Note that this construction is an instance of what Dietterich refers to as a funnel abstraction [40]: the value all possible robot configurations within the target free space get mapped to a single value: the value of that region. This is the basic abstraction from which the hierarchical MDP obtains its savings.

Examples of the NAMO MDP being solved in a simulation are shown in Fig. 34 and Fig. 35. The runtime of the NAMO MDP solver is linear in the number of obstacles for typical environments, rather than exponential as in the naive case [94, 95]. These results established the viability NAMO MDP solver in a physics-based world, but relied on several unrealistic assumptions about the accuracy of the model and of the force-control capabilities of the arms. Specifically, in these examples we simulated manipulation with a single 3DOF in-plane arm, and used a Kinodynamic-RRT (KD-RRT) to plan for the system [91]. KD-RRT is a general-purpose method for planning trajectories for arbitrary dynamical systems that works by sampling the raw control space of the robot. In our experiments we found that KD-RRT was not very efficient, and often produced low-quality plans when the



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

Figure 35: Execution example with more than 30 obstacles.

model was not known with high accuracy [95]. We address these issues in the next section.

6.3 Manipulation Control for Physics-Based Models

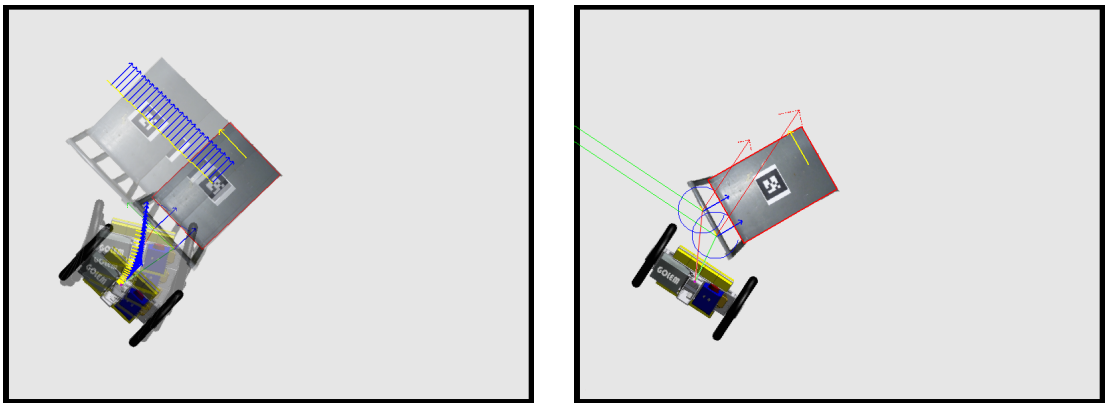
To perform manipulation actions such as the obstacle-clearing operations described above, we require an appropriate method for manipulating objects in the *PBRL* model space. The KD-RRT approach in Section 6.2.1 was a viable approach for object manipulation in simulation, but ignored several important aspects of (bi-manual) manipulation with physical robots which we address here. These considerations are relevant to any method for performing whole-body manipulation of objects on the ground.

Like many humanoid robot systems, we assume access to low-level current and velocity controllers of the individual wheels and joint motors, but not necessarily torque control[1]. Our goal is therefore to achieve velocity control of *grasped objects* in terms of velocities at the wheel motors and the joints in the body and arms.

Naively sampling this control space, as in the KD-RRT above, is not a viable option because it would result in out-of-plane movements of the grippers. Even assuming Cartesian velocity controllers for the grippers, sampling the gripper velocity space independently will cause the arms to fight each other, creating large grasp forces. Instead we require an approach that solves for the base and joint motor velocities directly as a function of the desired object velocity and the current body configuration. If done correctly, this will produce a coherent whole-body motion that manipulates the grasped object in the desired directions, without causing motion artifacts, internal forces, or manipulator singularities.

Constraints and Compliance The cart articulation controller in Chapter 4 provides a partial solution to this problem. This method worked by transforming the desired object velocity to the grippers, and feeding it through the manipulator Jacobian pseudo-inverse

[1]Force and torque is often difficult or impossible to control precisely; Many arms are not equipped with joint-torque sensors, and even given a force-torque sensor at the gripper, closed-loop force-control can be noisy and error-prone without accurate models of the dynamics of the manipulator.



(a) Expected Motion (b) Actual Motion with Constraint

Figure 36: Body control without considering constraints. Note large gripper forces from trajectory deviation, visualized in red (f_x), green (f_y), and blue (τ).

to obtain joint velocities. We then defined a set of useful velocities for the robot-object system, represented as key-frames (the motion primitives), which could be tracked by this controller.

There are two assumptions which must be relaxed for manipulating NAMO obstacles from the *PBRL* model-class with our robot. First, Chapter 4 was implemented on a PR2, which has a non-holonomic base. Krang has a differential-drive base, and therefore only has control in the x (forward-backward) and θ (rotational) directions. Second, the cart used in Chapter 4 was unconstrained, so the only concern in selecting manipulation velocities was the workspace of the arms (easily accomplished by restricting articulation to small displacements from the middle of the workspace).

To understand the importance of incorporating constraints into the body controller, consider what would happen if the robot tried to execute the trajectory in Fig. 36. Because of the fixed-wheels on the front of the cart, the cart would rotate, causing large rotational and translational forces to build up at the grippers (visualized as gripper force-torque vectors in Fig. 36(b)).

It may be tempting to rely on gripper compliance to compensate for the induced rotation, but there are two problems with this:

1. Active compliance biases the gripper velocities in proportion to perceived force, which quickly increases as objects deviate from their intended trajectories.

2. For planning we want to know where an object will end up when the robot tries to apply a velocity, but relying on compliance means we only find out when the action is executed on the physical system.

6.3.1 The Body Jacobian

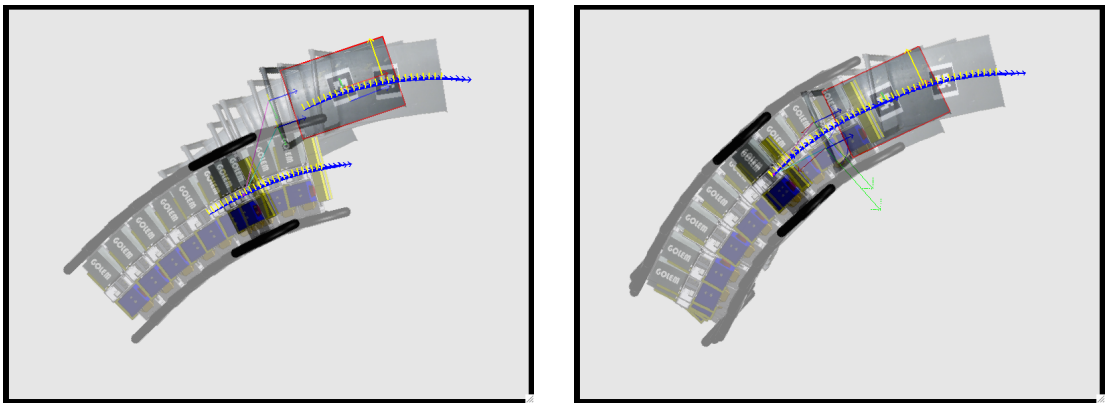
A more principled solution is to implement a whole-body Jacobian controller for the robot-object system, and simulating its outcome in the *PBRL* world for planning. A body Jacobian for this system is $8x3$, which is obtained by stacking three velocity transformations: one mapping base velocity in the robot frame to object velocity in the object-frame (with the base y-component dropped), and the others mapping gripper velocity in the robot frame to object velocity in the object frame (for Cartesian gripper controllers defined in the robot frame):

$$J = \begin{bmatrix} {}^{o}_{b}R^{T} & 0 & {}^{o}_{lg}R^{T} & 0 & {}^{o}_{rg}R^{T} & 0 \\ y_b - x_b & 1 & y_{lg} - x_{lg} & 1 & y_{rg} - x_{rg} & 1 \end{bmatrix}^{T} \tag{42}$$

where ${}^{A}_{B}R$ denotes the $2x2$ rotation from A to B for the robot base b, object o, left-gripper lg, and right-gripper rg. To use Eq. 42 for control the simplest method is to drop the base-y component (second row) to obtain the controllable Jacobian $\bar{J}$ and apply the pseudo-inverse to obtain body velocities from desired object velocities:

$$ {}_{r}v = \bar{J}^{+}{}_{o}v \tag{43}$$

A difficulty with this standard approach is that it divides the work evenly between the arms and the base. In practice we often want the base to do as much as possible, and leave the arms to do only what the base cannot (*e.g.* translation along base-y and rotation about arbitrary points). We can address this by projecting ideal base and gripper velocities into



(a) Without Steering (Infeasible) (b) With Steering

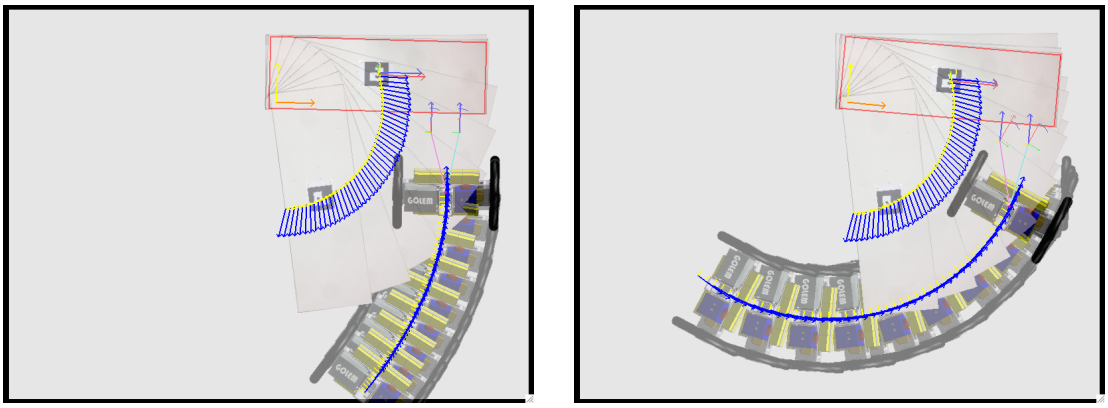
Figure 37: Comparison of body Jacobian with and without steering for cart manipulation.

the Jacobian pseudo-inverse nullspace:

$$ {}_{r}v = \bar{J}_r^{+}\,{}_{o}v + c_n(I - \bar{J}_r^{+}\bar{J}_r) \begin{bmatrix} {}_{b}v^*, \\ {}_{lg}v^*, \\ {}_{rg}v^* \end{bmatrix} \tag{44} $$

where the ideal base velocity is the desired object velocity transformed to the base frame (*i.e.* ${}_{b}v^* = {}^{b}_{o}\mathcal{Y}\,{}_{o}v$), the ideal gripper velocities ${}_{lg}v^*$ and ${}_{rg}v^*$ are zero, and c_n is a tunable gain. This method successfully biases the control effort to the base where possible, but failed to utilize the rotational DOF of the base to achieve lateral object velocities. As we will see, this can cause the arms to hit kinematic limits when generating longer manipulation trajectories.

Base Steering The Jacobian-based control approach described above offers a sound method for planning robot trajectories that are accurate and safe to execute, but we found that it frequently allowed the arms to drift out of the reachable workspace (Fig. 37(a) and Fig. 38(a)). This problem comes from having a differential drive base: any desired object velocity perpendicular to the wheels could only be achieved with the arms. Motivated by the nullspace controller in Eq. 44, we addressed this problem by splitting the Jacobian transform into two parts, and running a steering controller for the base at each intermediate waypoint before



(a) Without Steering (Infeasible)

(b) With Steering

Figure 38: Comparison of body Jacobian with and without steering for table manipulation.

simulating the gripper velocities. The process for each time-step can be summarized as follows:

1. Integrate the desired object-frame velocity to obtain an ideal object pose, assuming no constraints.

2. Transform the desired object velocity to the base and integrate it to obtain a desired base waypoint, and then apply a base steering controller to obtain a reachable base waypoint.

3. Compute the residual error between the object location at the reached base pose and the ideal object pose (*i.e.* the work to do be done by the arms), and apply a manipulation controller to minimize this error.

We implemented a threshold-based steering method for the differential-drive robot base. Let $e = \bar{s} - s$ denote the state-error vector for state s and reference (*i.e.* goal) $\bar{s}$, and $e_w = atan2(e_y, e_x) - s_\theta$ denote the waypoint error (*i.e.* the difference between the robot's current heading and the vector towards the current waypoint). Base steering is achieved by adding a term to the angular velocity that turns the robot to minimize the y error e_y, and zeroing the x velocity if this error was above a threshold:

$$e_\theta = \left\{ \begin{array}{ll} e_\theta, & \|e_{xy}\| \leq \varepsilon_{ang} \\ e_w, & \|e_{xy}\| > \varepsilon_{ang} \end{array} \right\} \tag{45}$$

$$e_{xy} = \left\{ \begin{array}{ll} e_{xy}, & e_w \leq \varepsilon_{lin} \\ 0, & e_w > \varepsilon_{lin} \end{array} \right\} \tag{46}$$

In our implementation $\varepsilon_{lin} = 0.01$ and $\varepsilon_{ang} = 0.05$. After applying the operations in Eq. 45 and Eq. 46 to the state-error vector, base controls can be computed using standard control laws (*e.g. PD*).

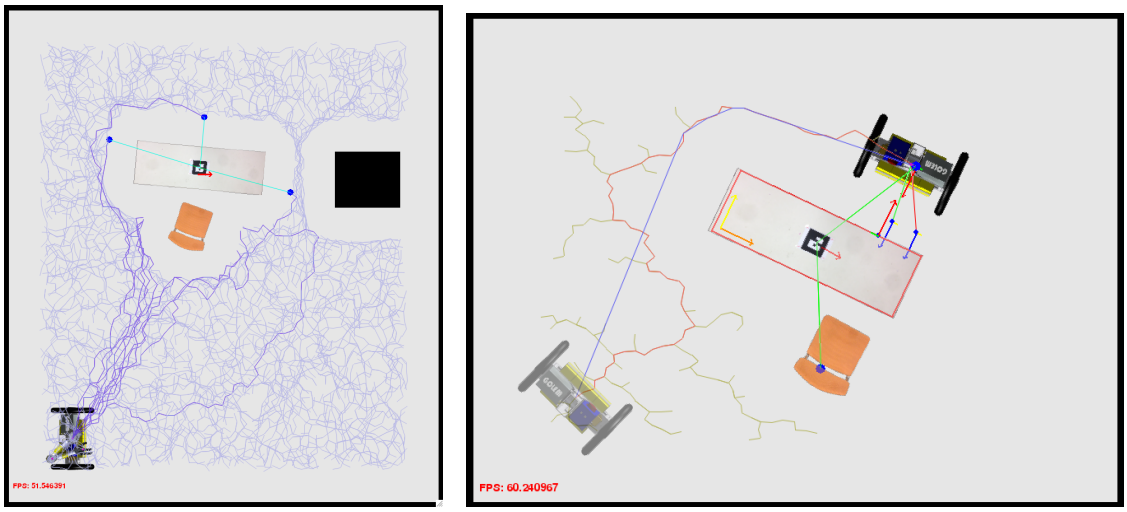
Incorporating the steering controller into the trajectory generation process relieved the burden on the arms to control errors in the y direction, allowing the grippers to remain in the middle of their workspace. Fig. 37 and Fig. 38 illustrate the effect of adding steering to the body controller.

This completes the description of the manipulation controller for *PBRL* models. In summary, we adopted a steered Jacobian-based controller to obtain robot base and gripper velocities as a function of desired object velocity, which we then integrated using the robot's current world beliefs to obtain executable trajectories.

6.3.2 Manipulation Policies

Using the method described in this section, the only free parameters for manipulation are (1) the grasp point on the object and (2) the desired object velocity. Manipulation planning therefore consists of searching over searching the space of valid grasp points and object velocities and evaluating the outcomes in the simulator. While far more compact than the raw control space, this still yields a large search space. For this reason, we defined a *model-dependent manipulation policy*, which indicated the set of achievable velocities and grasp points as a function of object dynamics. The policy we describe can be viewed as an optional planning heuristic which trades completeness for computation time.

We consider four model classes for the purposes of defining manipulation policies:



(a) Sampling for Unconstrained Table (b) After Grasping Locked Table

Figure 39: Sampling reachable grasp points depends on object dynamics. (a) For unconstrained object, points are sampled for each face and projected off the surface to leave room for grasp. (b) After grasping a *fixed-point* object. Note sampled boundary point and gripper offsets.

Model	**Velocities**	**Grasp Points**
Static	n/a	n/a
Unconstrained	Linear: $\pm v_x$ and $\pm v_y$ Angular: $\pm v_\theta$ at object center	Uniform
Anisotropic	Linear: $\pm v_u$ along unconstrained axis ($u = \arg\min_{x,y}(\mu_x, \mu_y)$. Angular: $\pm v_\theta$ at constraint anchor	Faces perpendicular to unconstrained axis.
Fixed-Point	Linear: None. Angular: $\pm v_\theta$ at constraint anchor	Top and bottom faces opposite to constraint

Figure 40: Manipulation policy for each model class.

static, *unconstrained*, *anisotropic*, and *fixed-point*. Static objects have sufficiently large values for all friction coefficients that no manipulation is possible. If an object is known to be static, the manipulation policy is null. Unconstrained objects have no physical constraints, and are free to move in any direction. For unconstrained objects we consider the three planar DOF separately, as in Chapter 3, for a total of six directions: $\pm\{x, y, \theta\}$. Anisotropic objects have a single large friction coefficient $\mu_x | \mu_y > 0.3$, and behave like wheeled bodies. To manipulate these objects we only consider angular velocities at the constraint and linear velocities along the unconstrained axis (*e.g.* if $\mu_x < 0.3$ and $\mu_y > 0.3$ then we only consider linear velocities along the constraint's x-axis). Fixed-point objects have either a distance joint or large values for both linear friction coefficient $\mu_x, \mu_y > 0.3$, and rotate about the joint anchor. These objects are the most constrained, and to manipulate them we consider only angular velocities $\pm v_\theta$ at the constraint anchor[2].

The grasp space is parameterized by angle from the object center-of-mass. Candidate grasp points are obtained by sampling an angle from a model-specific distribution over angles from the object center, computing the boundary point at this angle, projecting the boundary point off the object face (to leave room for the arms to grasp the object), and checking for a collision free navigation path to this point. If a path is found, this point is added to a list of viable grasp points. After the time-budget has been exceeded for this operation, the robot selects the shortest path point from the list of candidates. This process can be visualized in Fig. 39. Note that in Fig. 39(a) the object is unconstrained, and the grasp points are sampled from all object faces. In Fig. 39(b), however, the object has a locked wheel, and the grasp angles are constrained to the top and bottom face along the opposite end of the object[3].

These rules were sufficient for our purposes, but learning model-dependent policies

[2]Recall that for learning and planning object velocities are defined at the object center-of-mass, so these velocities will in general have non-zero linear terms when transformed to the object COM.

[3]Formally, we selected angles from the set $\pm atan2(\frac{\sqrt{3}}{2}, \frac{w}{2h})$, where w, h denote object width and height, respectively.

Subtask	Parameters
x, y = GetGraspPoint(o_i)	o_i: target object index
NavigateToPoint(x, y)	x, y: navigation goal point
Grasp(o_i)	o_i: target object index
ClearObstacle(f_i)	f_i: target free-space region index
Release(o_i)	o_i: target object index

Figure 41: Subtasks involved in the execution of a NAMO action.

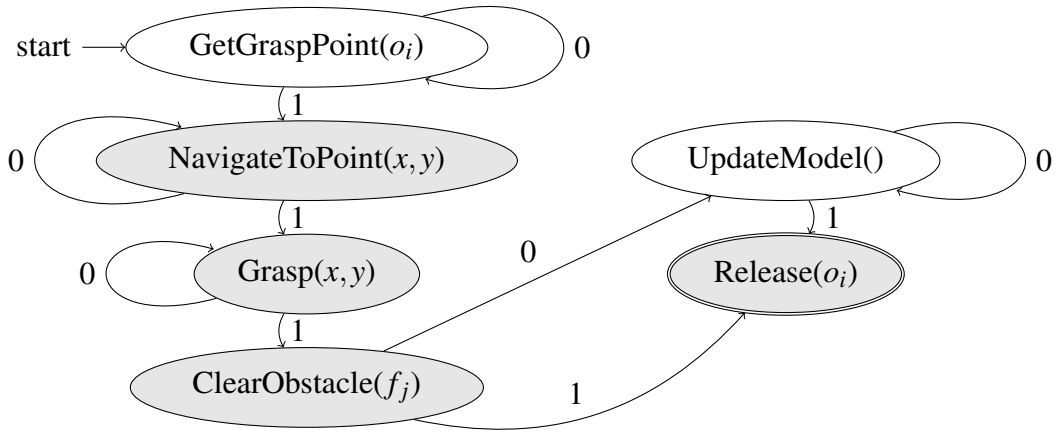


Figure 42: State-machine for performing a NAMO obstacle-clearing action, with predictive execution monitoring and model updating. Shaded nodes generate actual robot motion, and non-shaded nodes are purely computational. If triggered, *UpdateModel* operates on data recorded during the latest call to *ClearObstacle*.

(*i.e.* $P(a|\Phi, s)$) is an interesting area of research that may further justify our choice of model representation over more generic alternatives.

6.4 Predictive Execution

To solve a NAMO problem in realistic environments, the next step is designing a robust method for executing these manipulation actions on a physical robot. Our considerations include all of the safety and error-recovery issues from Chapter 4, and we introduce three new features for the NAMO problem:

1. NAMO requires sequencing multiple object interactions, and we therefore must consider navigating to, grasping, and releasing objects.

2. The termination condition is not a specific goal configuration, but rather *any* state that creates an opening.

3. Models may be wrong, so the robot should maintain expectations about the results of its actions, and abort to update its world model as necessary.

For these reasons a single high-level NAMO action is actually composed of a set of distinct intermediate subtasks: finding a valid grasp point according to the world state and model (Section 6.3.2), navigating to the grasp point, grasping the object, and executing a manipulation trajectory while (a) comparing forces with expected values from planning, aborting and updating the model as necessary, and (b) periodically checking for openings created using a path planner.

To handle this we implement a (stochastic) state machine for coordinating the execution of these operations. An abstract NAMO action instantiates each of the operations above with the appropriate parameters, as defined in Table 6.4.

The most important aspect of this construction is the mechanism for execution monitoring and model updating. We achieve this ability by leveraging our physics-engine framework to compute and save gripper forces during manipulation planning[4]. If unexpected forces are observed during execution, the subtask pauses and the data gathered during the aborted episode is passed to a learning procedure based on the method introduced in Chapter 5. After the model has been updated the planner releases the object and recomputes the NAMO policy for the current world state (note that the aborted action may have changed the action cost, and possibly even the free-space connectivity). A diagram illustrating the state machine for implementing a NAMO action is shown in Fig. 42.

As illustrated in Fig. 31, the *ClearObstacle* subtask uses the body controller defined in Section 6.3.1 for object manipulation. During manipulation, this subtask periodically checks for openings to the target free-space region using a fast circular-footprint RRT path

[4]This is similar to the idea of efference-copy in biological systems [185].

planner. For each timestep, *ClearObstacle* also computes the object-frame forces from the wrist sensor signals, and compares them to the expected forces applied during planning. This subtask terminates when either (a) a force is perceived that exceeds the maximum expected value by a user-defined threshold (150% in our implementation), or (b) the controller stalls or reaches a timeout.

6.5 Evaluation

In this section we present results demonstrating the performance of our full system on NAMO tasks with multiple unknown objects. The primary goal for these experiments is to demonstrate the adaptability conferred by *PBRL* model learning in an actual mobile manipulation task. In addition to replicating the model-identification results from Chapter 5, these experiments will also show how manipulation control can leverage the learned parameters explicitly for control. Finally, these results constitute the first real-robot implementation of the NAMO MDP.

All experiments were conducted in an office-like setting within the Humanoids lab at Georgia Institute of Technology. Task configurations were chosen to replicate the simulation result in Fig. 34, and then extend it using the new model-dependent manipulation policy from this chapter. We selected two tables to use as obstacles in this task: one which is square with actual mass $33Kg$, and another which is rectangular with mass $38Kg$. For these experiments we used a six-camera overhead vision system that tracked the robot and objects using AR-tags [47]. Both tables have four lockable casters which are difficult to perceive visually, even for humans.

Computation was distributed across three machines in these experiments. The core Krang controllers were run on-board using an Intel Core2 processor with a real-time linux operating system. A second desktop machine with an Intel 4-core i5 processor was dedicated to vision. All planning and state-machine execution was driven by a 2.6 Ghz i7 Macbook Pro laptop.

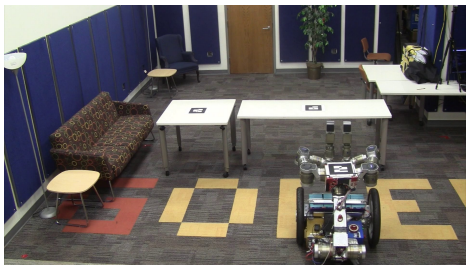
6.5.1 NAMO with a Static Constraint

In our first experiment we replicate the two-obstacle task from Fig. 34, in which the robot must update its model after encountering an immovable obstacle. Fig. 43 shows key-frames of the planning and execution trace of this task. The robot identified two free space regions and selected the square table as the lowest-cost obstacle to clear. When the robot grasped and attempted to push this object out of the way it encountered unexpected force in the $-x$ direction (see Fig. 43(d)), and halted to update its model. Using the method from Chapter 5, the robot estimated an anisotropic friction constraint in the middle of the table with two large coefficients, effectively rendering it static. The robot then recomputed the object Q-values and elected to pull the rectangular table down to create an opening. This action was successful, clearing the path shown in Fig. 43(h).

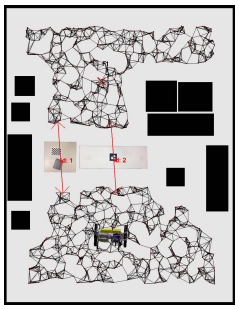
6.5.2 NAMO with a Non-Static Constraint

In the previous experiment the robot successfully identified a static constraint on the square table and re-planned accordingly. While an example of adaptive behavior, this result does not fully showcase the physical reasoning capabilities of a *PBRL* agent – a simple boolean model with a "movable" thresholded would have been capable of the same result. Thus, the only real advantage of modeling physics in experiment 1 was to provide a principled way detect when the robot's beliefs about the square table were wrong, and to stop pulling.

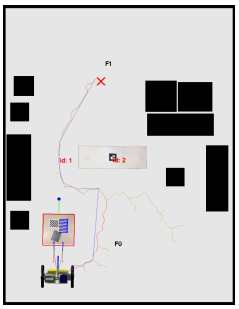
We therefore ran a second experiment in which one of the obstacles was movable, but only given knowledge of the physical constraint. Key-frames from this task are shown in Fig. 44. Planning and execution in this task follows the events in experiment 1, up to the point at which the robot attempts to pull the rectangular table. Instead of succeeding on its first attempt, the table rotates unexpectedly, due to a locked wheel on the left corner of the table. Rather than aborting, the robot the successfully estimates the coefficients and pose of this constraint, and executes a parameterized fixed-point action to rotate the table about this point (Section 6.3.2). The result of this rotation action can be seen in Fig. 44(j). This



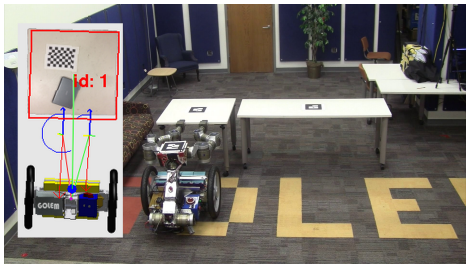
(a) Starting Configuration



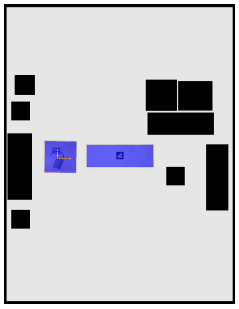
(b) Initial NAMO MDP



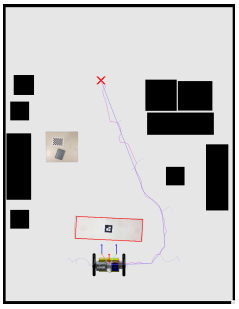
(c) Expected Solution: Pull Square Table



(d) Table is Stuck

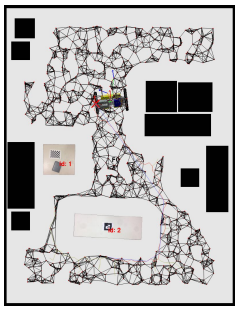


(e) Learns Static Constraint

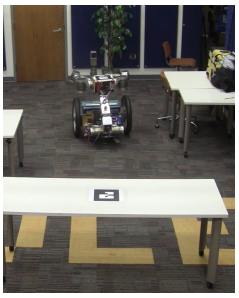


(f) New Solution: Pull Long Table

(g) Table Pulled Successfully



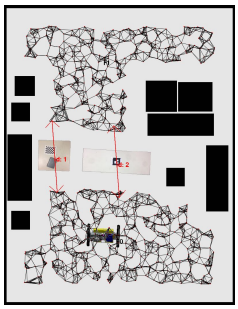
(h) Opening Found!



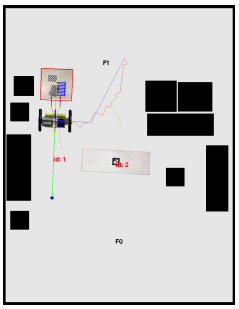
(i) Task Complete

Figure 43: Key-frames from Navigation Among Movable Obstacles task using physics-based model prior (PBRL). The square table has two locked casters, which is initially unknown to the robot.

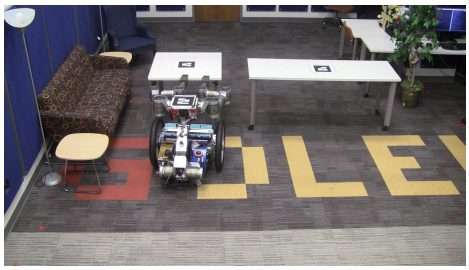
(a) Starting Configuration



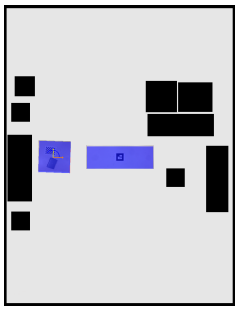
(b) Initial NAMO MDP



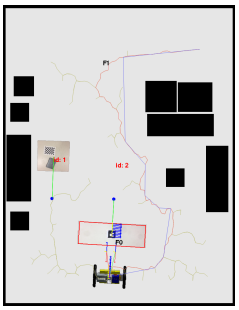
(c) Expected Solution: Push Square Table



(d) Table is Stuck

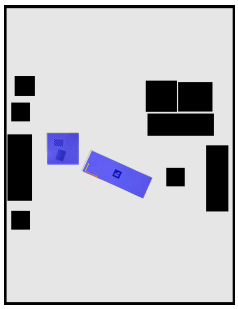


(e) Learns Static Constraint

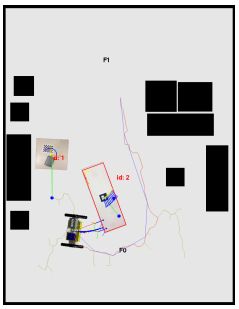


(f) New Solution: Pull Long Table

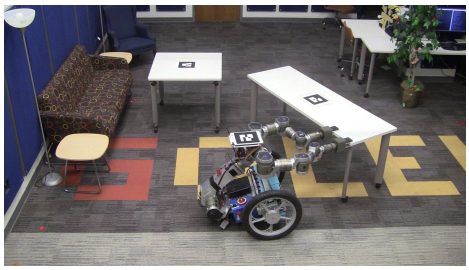
(g) Table Rotates Unexpectedly



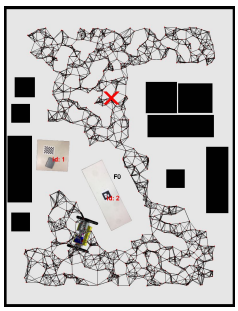
(h) Learns Parameters of Locked Caster



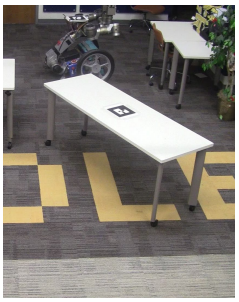
(i) New Solution: Rotate Long Table



(j) Table Rotated Successfully



(k) Opening Found!



(l) Task Complete

Figure 44: Key-frames from Navigation Among Movable Obstacles task using physics-based model prior (PBRL). The square table has two locked casters, and the rectangular table has a single locked wheel on the left side. Both of these properties are initially unknown to the robot.

	Experiment 1	Experiment 2
Number of Free-space Regions	2	2
Overall Runtime	4m 48s	7m 46s
Overall Planning time	0m 37s	0m 52s
Number of calls to NAMO planner	1	2
Number of calls to UpdateModel	1	2

Figure 45: Planning and execution statistics for NAMO experiments. Note that the NAMO planner was called twice in experiment 2 because the map changed during the first manipulation attempt on the rectangular table, which invalidated the action Q-values.

action successfully opened a path to the goal, as can be seen in Fig. 44(k). A table with relevant statistics comparing these two experiments can be found in Table 45.

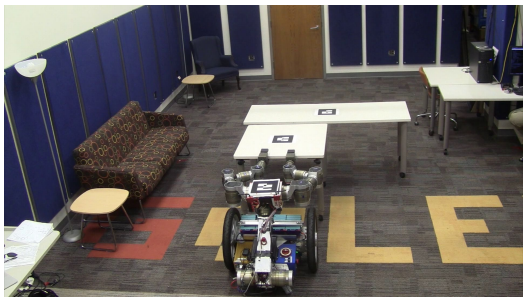
6.5.3 Learning Through Contact

So far the primary motivation for using a full multi-body simulation engine, as opposed to a set of individual object models, has been parsimony and compactness. The simulator's API gave a compact parameter space, and included the collision detection machinery needed for path planning. However, one of the primary functions of a physics engine is simulating the behavior of multiple *interacting* bodies. Can a *PBRL* agent take advantage of this functionality in its world model?

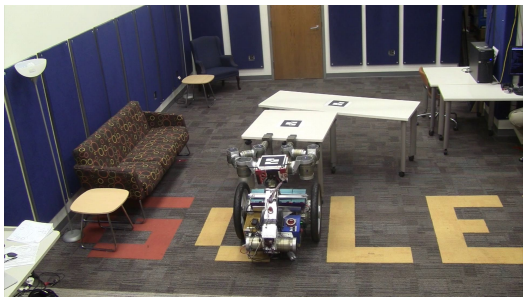
There are many mobile manipulation tasks that involve collision *prediction* (*e.g.* the Darpa Robotics Challenge [127]), but generalizing NAMO to the collision case is beyond the scope of this thesis. Instead, as this thesis is focused on *learning*, we ask a complementary question: Can the simulator enable the agent to learn through observing colliding bodies without ever directly measuring forces on one of the bodies directly?

Specifically, this experiment examines whether it is possible to estimate anisotropic friction parameters on an object through indirect contact. We expect this to be more difficult than the rigid grasp case for several reasons:

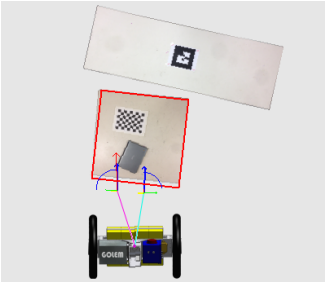
1. We can no longer directly compute force on the target object; Instead force is transmitted through a chain of intermediate calculations involving approximate collision resolution machinery.



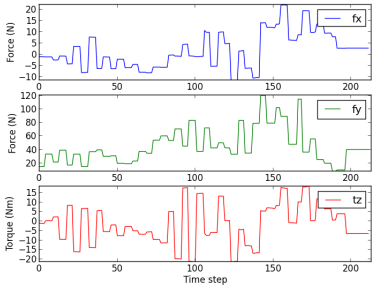
(a) Starting Configuration



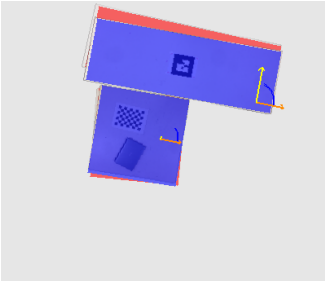
(b) Ending Configuration



(c) Manipulation Result (Simulation)



(d) Applied Force-Torque Data (COM)



(e) Learned Constraint on Long Table

Figure 46: Reasoning through contact: Robot successfully estimates parameters of a locked wheel on the rectangular table despite never touching it directly.

2. Collision simulation depends on additional model parameters (restitution and surface friction).

3. Collision simulation is critically sensitive to object geometry.

4. The contact point is crucial for determining how both objects move, and error in vision can introduce significant bias in the transmission of forces.

Despite these potential difficulties, Fig. 46 shows preliminary results which suggest that it is possible to estimate anisotropic friction parameters on an object through indirect contact. In this experiment, the robot grasped a square table and pushed it into a rectangular table which had a locked caster on the lower-right corner. Fig. 46(e) shows the final model estimated by the robot for this system, which contains an anisotropic friction joint on the rectangular table close to its true location. Although we have not formally assessed the tolerance of the fixed-point controller, the final pose of this constraint was within the range

of values that we found could be handled by gripper compliance.

Unfortunately due to hardware malfunctions, we were unable to obtain collision datasets across a broader range of objects and configurations. Therefore the accuracy and generality of this result has not yet been established. However, we find this result to be promising, and expect it may open the door to a range of human-like reasoning patterns.

6.5.4 Failure Cases

Although the results described here demonstrated adaptive behavior, there were several common points of failure in our experiments. Foremost, mobile manipulation tasks such as NAMO require the synthesis of many complex components, from low-level hardware to high-level perception and planning algorithms, and the brittleness of the overall system was one of the primary bottlenecks in this research. For example, precise calibration of the overhead vision system or the manipulator kinematics often made the difference between a grasp or manipulation succeeding and failing. In addition, many of the subroutines involved in our overall method involved some form of randomness, including PRM free-space detection, RRT path planning, grasp-point sampling, and MCMC model inference. We were able to tune these algorithms to achieve $90+\%$ success rates on each subroutine, but the long horizon of the tasks we consider here led to frequent failures at some point in the pipeline.

6.6 Discussion

The purpose of this chapter was to introduce methods for planning with *PBRL* models, and implement a framework which unified the methods discussed in this thesis. We began with an overview of the NAMO problem, and described a hierarchical MDP model for this task which grounded abstract dynamics uncertainty in terms of *PBRL* model beliefs. We then described a Jacobian-based control stack on our robot, Golem-Krang, for achieving whole-body manipulation of constrained bodies from the *PBRL* model class. These controllers and the associated logic for identifying valid grasp points and model parameters was embedded

in a state machine for coordinated, robust execution of NAMO obstacle-clearing actions. Finally, we considered the problem of planning on top of these abstract NAMO actions, and introduced an additional optimization in the form of a model-dependent manipulation policy for efficient planning with constrained objects.

Our experiments focused on demonstrating the feasibility of our methods on a manipulation problem that forced the robot to detect errors in its world model and adjust its plan accordingly. The NAMO task presented here included unobservable physical properties, in the form of lockable wheels, that required the robot to either switch objects (experiments 1 and 2), or switch control strategies (experiment 2).

Our early results in experiment 2 suggested that a model-specific manipulation policy is important for manipulation planning. Although it would have been possible for the robot to find a kinematically feasible trajectory through exhaustive search over grasp and velocity parameters, it required a time-consuming search through a control space $u_{body} \in \mathbb{R}^4$, requiring expensive trajectory evaluations. From these observations we would argue that the mapping from dynamics models to control is a useful and fundamental concept that should be explored in greater depth in the future.

Additional justification for this point of view can be found in motor neuroscience. Research suggests that humans are adept at learning context or object-specific control strategies [184]. Further, it has been proposed that the human motor system is modular, and that the core organizational structure involves *pairs* of forward (predictive) and inverse (control) models. In terms of this "*MOSIAC*" model, the manipulation policy we describe fills a necessary role as the mapping between the dynamics model and the appropriate controller. Following the terminology of Wolpert et al. [185], we have used Φ as a "context" parameter c to identify the appropriate controller. However, in our work the manipulation policy was hand-coded rather than learned. These results therefore motivate the question of how to learn a model-dependent manipulation policy empirically. Overall our experiments illustrated the viability of *PBRL* as a method for producing adaptive behavior in online mobile

manipulation scenarios.

CHAPTER VII

DISCUSSION AND FUTURE DIRECTIONS

At its core, *Physics-Based Reinforcement Learning* is about modeling the physical motion of *objects*, and how they influence one another. Many areas of physics do not center on objects, for example fluid dynamics, thermodynamics, and relativity. These dynamics may be relevant to robots operating in some circumstances, but do not feature prominently in the domestic manipulation tasks we consider. As a result, this thesis has taken the strict view that a *PBRL* agent uses a physics engine as a model space; however, there could be more general notions of what it means to be a physics-based agent.

In this chapter we discuss the assumptions behind *PBRL*, and tease apart our specific contributions from the broader implications of a physics-aware RL agent. We will also identify the limitations of our current implementation, and discuss several generalizations of our approach that seem promising given the current state of machine learning research.

7.1 Core Assumptions

Here we elaborate the specific assumptions involved in our implementation of a *PBRL* agent, as well as the more general assumptions implied in a *PBRL* approach.

Specific Assumptions The methods and results presented in this thesis were focused on manipulation of furniture-like objects with a mobile robot. We therefore focused on a subset of planar object dynamics that were useful for these problems. These assumptions can be summarized as follows:

- The world state could be parameterized as the union of the planar position and velocity of all objects in the domain, and actions could be summarized as the force and torque applied to a *single* target object.

- A 2D physics-engine could capture all relevant dynamical effects.
- There were only three necessary parameter classes: rigid body parameters to capture collisions, anisotropic friction parameters to model wheels, and distance constraint parameters to model revolute motion.
- The robot was equipped with force-torque sensors, and all forces applied to objects were transmitted through these sensors.
- Object geometry was known, and mass distributions could be computed assuming uniform density over polygonal object geometry.
- Object state could be obtained from an overhead-vision system using AR-tags, and velocity could be approximated by finite-differencing these pose estimates.
- The robot was equipped with an appropriate action space for the range of objects considered by the model space.

General Assumptions In addition to our specific implementation choices, there are several assumptions inherent to a physics-based approach that are not required of more abstract decision problems:

- The world dynamics can be fully described in terms of objects and their interactions.
- The choice of object state representation is motivated by an underlying differential equation capturing object motion, *e.g.* position and velocity.
- Object interactions are modeled using established physical laws, encoded in specific computational tools.
- The choice of model parameters is determined by the auxiliary variables needed to parametrize these physical laws.

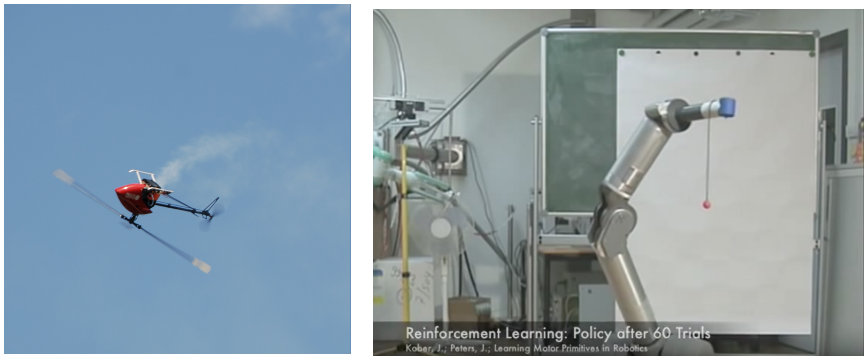
- These parameters directly influence the evolution of the world state, allowing a simulator to be used as a generative model for the observed state-action data.

- The agent can provide input to the system, and measure this input using a sensor (*e.g.* force sensor, pressure sensor), which (a) summarizes the input in some physically-defined unit (N, pascal, etc), and (b) the locus of input is known (*e.g.* grasp point, contact distribution, etc.)

7.2 Limitations

There are a number of limitations that follow from the assumptions discussed above. The most important practical consideration is the reliance on known geometric models and an object tracking system. Without a reliable solution to these problems for natural scenes, the methods presented here will be of little practical value. We also restricted our attention to a narrow set of furniture-like objects in two-dimensions. As a result we cannot solve dynamic manipulation problems in three dimensions, for example moving a book from a table to a shelf. Many pick-and-place tasks in 3D involve simple unconstrained rigid bodies, so our method would be more relevant for interacting with kinematically-constrained bodies such as doors, drawers, levers, and other jointed objects. To the extent which our implementation involved 3D gripper motion, the trajectories were defined by hand and not collision checked against 3D geometric models. Our NAMO manipulation actions required object-specific grasp and release trajectories for both arms. This is a less severe limitation, as there are existing grasp and motion-planning algorithms for solving these problems assuming object geometry is known. Our implementation also does not reason about non-object world properties, *e.g.* surface friction of carpet vs. hard-wood. This may be an important addition for deploying mobile manipulation systems in domestic environments with heterogeneous surfaces.

More generally, *PBRL* was not designed for problems that do not involve objects or



(a) Autonomous Helicopter (b) Ball-In-Cup

Figure 47: Example Reinforcement Learning problems in physical domains that are not ideal candidates for *PBRL* approach. (a) Autonomous helicopter trained with Reinforcement Learning method based on non-parametric regression model. (b) Ball-in-Cup task trained with model-free policy-search method.

physical agents, *e.g.* warehouse logistics, stock portfolio management, power grid management, or elevator scheduling, to name a few. Perhaps less obviously, there are also many physical decision problems that are not ideal candidates for a *PBRL* approach. For example, tasks involving specific systems with complex dynamics, such as a helicopter, rally car, or airplane, offer a distinct set of challenges in which multi-body dynamics is of little relevance. The acrobatic maneuvers depicted in Fig. 47(a) were achieved by a model-predictive-control strategy using a non-parametric regression model (LWR) on a position and velocity representation of the helicopter state [117]. *PBRL* is also not ideal for low-level control problems which are readily approached with model-free methods, such as the ball-in-cup task shown in Fig. 47(b) (solved using a policy-search approach, [77]).

7.3 Extending PBRL

As mentioned above, many planning and control problems in physical domains are outside the current scope of *PBRL* as presented here. In this section we define the basic requirements of a *PBRL* problem, and consider the steps involved in generalizing our method to a wider array of domains.

7.3.1 Considerations

There are two core considerations that determine whether a *PBRL* approach is appropriate for a given problem. The first is whether a known dynamics engine can encapsulate all relevant dynamics, and the second is whether an object-oriented state representation provides sufficient input to this dynamics engine. In problems involving fluid dynamics, for example, the state representation is no longer position and velocity vectors for rigid bodies but scalar and vector *fields* describing pressure and fluid flow. For this reason it is not straight-forward to generalize the methods considered here to problems involving interaction with fluids[1]. By contrast, generalizing from 2D to 3D is more straight-forward, in principle, because it involves the same basic approach: a search over the model space of a fully-parameterized engine.

7.3.2 Engine-Based Approach

The simplest approach to designing PBRL models, and the approach taken in this thesis, is to obtain a fully-functional physics engine that is appropriate for the target domain, and define a learning interface over its model parameters. At a high level this procedure can be summarized as follows:

1. Obtain an engine which can simulate the desired scene.
2. Define geometric models for the relevant objects.
3. Define a model space for each object using the engine's modeling API.
4. Assign appropriate priors either manually or with a probabilistic modeling tool such as PyMC [122]. Alternatively, define parameter bounds and implement or obtain a optimizer such L-BFGS [189].

[1] We note, however, that fluid physics is well understood, and the like the rigid-body case offers a number of ready computational tools.

5. Define an appropriate set of controllers and planners for the agent to interact with the domain.

6. Implement an RL agent interface for executing actions against the simulator and the real world, aggregating state-action data, and updating the world model as often as necessary[2].

While this approach is sound, the feasibility of inference on larger model spaces and 3D scenes is an open research question. Even for the restricted model space considered here, we had to do a considerable amount of tuning to achieve reasonable performance (see Section 5.3.2). One possible route to achieving more efficient inference algorithms is to utilize recent developments in fully differentiable contact physics models, *e.g.* [176], to allow gradient-based optimization [189] (for the maximum-likelihood case) or Hamiltonian MCMC [116] (for the MAP case).

7.3.3 Alternative Model Representations

Physics engines are only approximations of natural dynamics, and there may be problems which will require higher fidelity than existing simulation tools can provide. In this section we consider alternative approaches to encoding physical knowledge that do not involve a black-box engine. The approaches discussed are motivated by published results, but should be considered speculative.

Deep-Learning Approaches Neural networks are currently undergoing a renaissance in machine learning. Like non-parametric regression, deep-learning has the appealing capability of learning functions that we, as systems engineers, do not necessarily understand

[2]Recall that data-aggregation may involve non-trivial processing of sensory data to obtain a compact dataset in terms of the target objects rather than the robot. This is especially important when using Eulerian simulators that define constraints implicitly, as opposed to Lagrangian simulators in generalized coordinates, since they can be quite inaccurate at simulating kinematic structures such as robot arms.

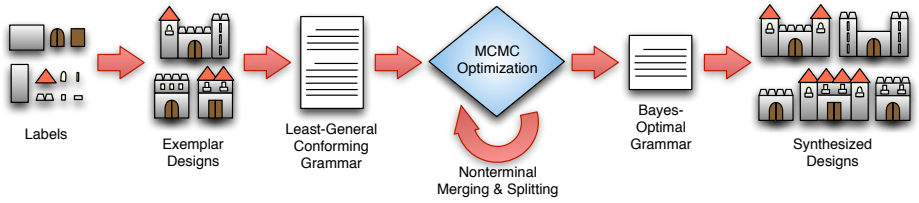


Figure 48: The pipeline for learning design patterns with grammar induction, reproduced from [168]. A set of example designs, each comprising a hierarchy of labeled components, are used to produce an initial, specific grammar. Then, Markov chain Monte Carlo optimization is employed to explore a space of more general grammars, balancing the descriptive power of each one against its representation complexity. At the end of this process, the best scoring grammar is returned, and can be sampled to generate new designs.

or have models for (at the expense of transparency). There is already evidence that deep-networks have the capacity to simultaneously capture a variety of rigid and non-rigid dynamical effects. In [128], a rectified linear unit (ReLU) neural network was used to directly model the nonlinear dynamics of a helicopter using a training set that included acrobatic maneuvers (as an alternative to LWR, discussed in Section 7.2). Deep neural networks have also been used to model partial differential equations (PDEs) for segmentation in vision [99]. Therefore an interesting possibility is to attempt to encode the mathematical principles supporting rigid body simulation (*i.e.* Newton-Euler integration, differential constraints, and collisions), in a neural architecture[3]. To-date there is no research which has shown how collision dynamics can be computed neurally; however, if successful, this approach could provide a more flexible approach to modeling physical dynamics that retains the favorable sample efficiency of the engine-based approach.

Probabilistic Differential Grammars The Deep-Learning approach discussed previously constitutes a significant departure from the standard computational model for implementing dynamics algorithms. Another more familiar approach to eliminating the need

[3]This can be motivated by analogy to the well-known convolutional neural network, which incorporated the vision-related concepts of local receptive fields and retinotopic invariance via the convolution operation.

for a black-box physics engine is to attempt to assemble the engine from parts. Probabilistic grammars offer a mechanism for achieving this by allowing data-driving composition of arbitrary structured components. [168] shows example of probabilistic grammar induction for learning generative hierarchical structure in images, 3D structures (Sakura trees, space ship models), and websites. An illustration of this process can be seen in Fig. 48. Grammar inference generalizes the engine-based approach by permitting search over the structural components of the physcs-engine code itself. The key advantage of a grammar-based approach is that it can leverage complex algorithms, *e.g.* collision detectors, in their entirety while permitting a search over the simpler components. This offers an appealing middle-ground between tightly structured approaches, such as placing a prior on a complete engine, and loosely structured approaches such as a neural network model or non-parametric regression. The drawback to this approach is that it can eliminate the optimizations present in production simulation software, such as broad-phase collision detection in which the engine maintains a binary search tree over bounding boxes to reduce the naive $O(n^2)$ complexity of overlap testing.

The approaches discussed in this section offer greater modeling power within the *PBRL* framework. However, this expressiveness necessarily comes at the cost of having to search a larger parameter space. In order to be feasible for online applications, our goal will be to find the right balance between overly precise physical models which are brittle and hard to fit, and coarse models that lack expressive power.

CHAPTER VIII

CONCLUSION

The central challenge we consider in this thesis is the need for *learnable* models of multi-body object dynamics. These models are integral to many of the planning and control algorithms driving recent successes in modern robot manipulation. In this thesis we argued that equipping robots with methods for estimating their own simulation models is an important step towards achieving these successes in the real world.

This thesis presented a coherent framework called *Physics-Based Reinforcement Learning* to meet this challenge. PBRL is a model-based Reinforcement Learning framework which uses a physics engine as the core model representation. We presented a set of manipulation planning methods designed to handle the *PBRL* model space. We also introduced two manipulation tasks that are well modeled as RL problems. The first was confined to a table-top setting, and was driven by an abstract cost function measuring table cleanliness. For this task we presented a new algorithm called the task-space RRT that was able to directly search the provided cost function using learned action models. Although demonstrated with fairly crude action primitives, the TS-RRT is equally relevant to *mobile* manipulation problems, and can operate with the continuous action space defined in later chapters. The second task was Navigation Among Movable Obstacles. We presented the first NAMO implementation which was able to operate in an underspecified environment, and adapt to a range of common objects. To the best of our knowledge, this is the first time this type of behavior has been achieved in a mobile manipulation setting.

While the goal of *PBRL* is to be parsimonious, the results presented here utilize only a small fraction of the modeling power of modern simulation tools. *PBRL* makes strong parametric assumptions that were appropriate for NAMO-like tasks, but will be insufficient

for many other mobile manipulation problems with different types of objects. However, we have yet to exhaust the modeling potential of 2D rigid body simulation tools, and there are many additional effects that would be useful to incorporate into the *PBRL* model space, such as elastic motion, fluid and non-rigid motion, and three-dimensional motion. These dynamics are ubiquitous in natural environments, and uncertainty of this kind will be the norm when deploying robots outside of factories and laboratories.

Some may also argue that our parametric approach also disregards the progress made in model-based RL with more flexible non-parametric models. In fact, we view these approaches as complementary rather than contradictory. Indeed one of the most interesting directions of future work in this area is unifying parametric and nonparametric methods. One simple approach would be to replace the Gaussian likelihood in Eq. 25 with a Gaussian Process (*i.e.* "wrap" a GP around the *PBRL* prediction). In this scheme, *PBRL* parameters could be sampled via MCMC, or from another posterior approximation, and the final prediction would be obtained by evaluating the GP with this mean. Such an approach would inherit the advantages of GP models for handling arbitrary dynamics and explicitly representing uncertainty, while also incorporating differential effects and collision machinery to help the model extrapolate. Ultimately, we contend that some sort of physics engine is necessary to encapsulate what is known about the dynamics (and appearance) of objects in the world, but that this structure will likely need to be augmented with methods to soften its parametric assumptions.

Model richness and policy richness also go hand in hand: as we add model flexibility we often require additional control methods to handle these cases. The manipulation methods presented in Chapter 6 were sufficient for planar object manipulation with anisotropic friction constraints, but will need to be generalized as we introduce additional object models. Creating appropriate controllers or policy representations for arbitrary objects is a non-trivial problem which will likely occupy researchers in robotics for decades.

Another crucial omission is visual perception. All methods presented here assumed the

geometry of the world was known. This is clearly unrealistic for natural tasks. Our model is in the tradition of generative models of object dynamics [137], and we are very sympathetic with efforts to build generative models of object appearance that are currently emerging in the vision literature [82].

More generally, *PBRL* can be viewed as an ontological constraint on the world model: it is governed by the laws of physics. We hope that this approach helps to close the representational gap between the sorts of models used in Reinforcement Learning and the models that robotics engineers use in practice. If successful, this approach may yield opportunities for learning representations that are currently engineered by hand in robotics. We feel that rich generative models of this sort stand to address some of the deepest challenges in robotics and artificial intelligence, and allow future research to advance to higher levels of reasoning and behavior.

REFERENCES

[1] "Energy Information Administration a look at the u.s. commercial building stock: Results from eia's 2012 cbecs." http://www.eia.gov/consumption/commercial/reports/2012/. Accessed: 2015-07-31.

[2] "Energy Information Administration total square feet of u.s. housing units by census region and type of housing unit, 1980-2005." http://www.eia.gov/emeu/efficiency/recs_4_table.pdf. Accessed: 2015-07-31.

[3] AKELLA, S. and MASON, M., "Posing polygonal objects in the plane by pushing," *The International Journal of Robotics Research*, vol. 17, no. 1, p. 70, 1998.

[4] AN, C., ATKESON, C., and HOLLERBACH, J., "Estimation of inertial parameters of rigid body links of manipulators," in *Decision and Control, 1985 24th IEEE Conference on*, vol. 24, IEEE, 1985.

[5] ARGALL, B., CHERNOVA, S., VELOSO, M., and BROWNING, B., "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[6] ATKESON, C., AN, C., and HOLLERBACH, J., "Estimation of inertial parameters of manipulator loads and links," *The International Journal of Robotics Research*, vol. 5, no. 3, 1986.

[7] ATKESON, C., "Using locally weighted regression for robot learning," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 958–963, IEEE, 1991.

[8] ATKESON, C., MOORE, A., and SCHAAL, S., "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, no. 1, 1997.

[9] ATKESON, C. and SCHAAL, S., "Robot learning from demonstration," in *Machine Learning-International Workshop then conference*, pp. 12–20, Citeseer, 1997.

[10] BATTAGLIA, P. W., HAMRICK, J. B., and TENENBAUM, J. B., "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18327–18332, 2013.

[11] BECEDAS, J., TRAPERO, J., SIRA-RAMIREZ, H., and FELIU-BATTLE, V., "Fast identification method to control a flexible manipulator with parameter uncertainties," in *ICRA 2007*, IEEE, 2007.

[12] BELLMAN, R., "Dynamic programming and lagrange multipliers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, no. 10, p. 767, 1956.

[13] BENTIVEGNA, D. and ATKESON, C., "Learning from observation using primitives," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1988–1993, Citeseer, 2001.

[14] BERTSEKAS, D. P., BERTSEKAS, D. P., BERTSEKAS, D. P., and BERTSEKAS, D. P., *Dynamic programming and optimal control*, vol. 1. Athena Scientific Belmont, MA, 1995.

[15] BHAT, K., SEITZ, S., POPOVIĆ, J., and KHOSLA, P., "Computing the physical parameters of rigid-body motion from video," *ECCV*, 2002.

[16] BHAT, K., TWIGG, C., HODGINS, J., KHOSLA, P., POPOVIĆ, Z., and SEITZ, S., "Estimating cloth simulation parameters from video," in *SIGGRAPH*, Eurographics Association, 2003.

[17] BILLINGS, S. A., *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[18] BISHOP, C. M. and OTHERS, *Pattern recognition and machine learning*, vol. 1. springer New York, 2006.

[19] BOUTILIER, C., DEARDEN, R., and GOLDSZMIDT, M., "Stochastic dynamic programming with factored representations," *Artificial Intelligence*, vol. 121, no. 1, 2000.

[20] BOYAN, J. and MOORE, A. W., "Generalization in reinforcement learning: Safely approximating the value function," *Advances in neural information processing systems*, pp. 369–376, 1995.

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

[22] BROOKS, S., GELMAN, A., JONES, G., and MENG, X.-L., *Handbook of Markov Chain Monte Carlo*. Taylor & Francis US, 2011.

[23] BROWN, S. and SAMMUT, C., "An architecture for tool use and learning in robots," in *Australian Conference on Robotics and Automation*, Citeseer, 2007.

[24] CAREY, S. and SPELKE, E., "Domain-specific knowledge and conceptual change," *Mapping the mind: Domain specificity in cognition and culture*, pp. 169–200, 1994.

[25] CATTO, E., "pybox2d." http://code.google.com/p/pybox2d/, June 2012.

[26] CATTO, E., "pybox2d." http://code.google.com/p/pybox2d/, June 2012.

[27] CETIN, A. and ADLI, M., "Cooperative control of a human and a robot manipulator for positioning a cart on a frictionless plane," *Mechatronics*, vol. 16, no. 8, pp. 461–469, 2006.

[28] CHENG, P., FINK, J., and KUMAR, V., "Abstractions and Algorithms for Cooperative Multiple Robot Planar Manipulation," *Robotics: Science and Systems IV*, p. 143, 2009.

[29] COHEN, B., CHITTA, S., and LIKHACHEV, M., "Search-based planning for manipulation with motion primitives," in *IEEE International Conference on Robotics and Automation*, (Anchorage, Alaska), 2010.

[30] CRAIG, J. J., *Introduction to robotics: mechanics and control*. Pearson/Prentice Hall Upper Saddle River, NJ, USA:, 2005.

[31] DAVIS, E. and MARCUS, G., "The scope and limits of simulation in cognition and automated reasoning," *Under submission*, 2013.

[32] DEAN, T., KAELBLING, L., KIRMAN, J., and NICHOLSON, A., "Planning with deadlines in stochastic domains," in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 574–579, Citeseer, 1993.

[33] DEGRIS, T., SIGAUD, O., and WUILLEMIN, P., "Learning the structure of factored markov decision processes in reinforcement learning problems," in *ICML*, ACM, 2006.

[34] DEISENROTH, M., NEUMANN, G., and PETERS, J., "A survey on policy search for robotics," *Foundations and Trends in Robotics*, 2013.

[35] DEISENROTH, M. and RASMUSSEN, C., "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011.

[36] DEISENROTH, M., RASMUSSEN, C., and FOX, D., "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[37] DEISENROTH, M., RASMUSSEN, C. E., and PETERS, J., "Model-based reinforcement learning with continuous states and actions," 2008.

[38] DEMAINE, E., DEMAINE, M., and O'ROURKE, J., "Pushpush and push-1 are np-hard in 2d," *arXiv preprint cs/0007021*, 2000.

[39] DIANKOV, R. and KUFFNER, J., "Randomized statistical path planning," in *Proceedings of IEEE/RSJ 2007 International Conference on Robots and Systems (IROS)*, Citeseer, 2007.

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

[41] DIETZ, L., "Directed damnr graph notation for generative models," *Max Planck Institute for Informatics, Tech. Rep*, 2010.

[42] DILLMAN, B. and STEINHAUS, P., "ARMAR II–a learning and cooperative multimodal humanoid robot system," *International Journal of Humanoid Robotics*, vol. 1, no. 1, pp. 143–155, 2004.

[43] DIUK, C., COHEN, A., and LITTMAN, M., "An object-oriented representation for efficient reinforcement learning," in *ICML*, ACM, 2008.

[44] DOSHI-VELEZ, F., WINGATE, D., ROY, N., and TENENBAUM, J., "Nonparametric bayesian policy priors for reinforcement learning," 2010.

[45] DUFF, D., WYATT, J., and STOLKIN, R., "Motion estimation using physical simulation," in *ICRA*, IEEE, 2010.

[46] FAN, H., ZIPF, A., and FU, Q., "Estimation of building types on openstreetmap based on urban morphology analysis," in *Connecting a Digital Europe Through Location and Place*, pp. 19–35, Springer, 2014.

[47] FIALA, M., "Artag, a fiducial marker system using digital techniques," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 590–596, IEEE, 2005.

[48] FOX, D., "KLD-sampling: Adaptive particle filters and mobile robot localization," *Advances in Neural Information Processing Systems (NIPS)*, pp. 26–32, 2001.

[49] FRAZZOLI, E., DAHLEH, M., and FERON, E., "Robust hybrid control for autonomous vehicle motion planning," in *Proceedings of the 39th IEEE Conference on Decision and Control, 2000*, vol. 1, 2000.

[50] GERKEY, B., VAUGHAN, R., and HOWARD, A., "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, pp. 317–323, Citeseer, 2003.

[51] GISZTER, S., MUSSA-IVALDI, F., and BIZZI, E., "Convergent force fields organized in the frog's spinal cord," *Journal of Neuroscience*, vol. 13, no. 2, pp. 467–491, 1993.

[52] GLASSMAN, E. and TEDRAKE, R., "LQR-Based Heuristics for Rapidly Exploring State Space," *Submitted to ICRA*, vol. 2009, 2010.

[53] GOEBEL, R., SANFELICE, R. G., and TEEL, A., "Hybrid dynamical systems," *Control Systems, IEEE*, vol. 29, no. 2, pp. 28–93, 2009.

[54] GREEN, P. J., "Trans-dimensional markov chain monte carlo," *Oxford Statistical Science Series*, 2003.

[55] GUESTRIN, C., KOLLER, D., GEARHART, C., and KANODIA, N., "Generalizing plans to new environments in relational mdps," in *In International Joint Conference on Artificial Intelligence (IJCAI-03*, Citeseer, 2003.

[56] GUESTRIN, C., KOLLER, D., PARR, R., and VENKATARAMAN, S., "Efficient solution algorithms for factored mdps," *Journal of Artificial Intelligence Research*, pp. 399–468, 2003.

[57] HAMRICK, J., BATTAGLIA, P., and TENENBAUM, J., "Internal physics models guide probabilistic judgments about object dynamics," in *Proceedings of the 33rd annual conference of the cognitive science society*, 2011.

[58] HART, P. E., NILSSON, N. J., and RAPHAEL, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[59] HARUNO, M., WOLPERT, D. H., and KAWATO, M., "Mosaic model for sensorimotor learning and control," *Neural computation*, vol. 13, no. 10, pp. 2201–2220, 2001.

[60] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., and FRANKLIN, J., "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[61] HAUSER, K., BRETL, T., HARADA, K., and LATOMBE, J., "Using motion primitives in probabilistic sample-based planning for humanoid robots," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Springer, 2006.

[62] HAUSER, K., NG-THOW-HING, V., and GONZALEZ-BAÑOS, H., "Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task," in *Proc. of the Int'l Symposium on Robotics Research (ISRR)*, Citeseer, 2007.

[63] HERMANS, T., LI, F., REHG, J. M., and BOBICK, A. F., "Learning Contact Locations for Pushing and Orienting Unknown Objects," in *IEEE-RAS International Conference on Humanoid Robotics*, October 2013.

[64] HOEY, J., ST-AUBIN, R., HU, A., and BOUTILIER, C., "Spudd: Stochastic planning using decision diagrams," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1999.

[65] IAGNEMMA, K., GENOT, F., and DUBOWSKY, S., "Rapid physics-based rough-terrain rover planning with sensor and control uncertainty," in *ICRA 1999*, vol. 3, IEEE, 1999.

[66] IMAMIZU, H., MIYAUCHI, S., TAMADA, T., SASAKI, Y., TAKINO, R., PUTZ, B., YOSHIOKA, T., and KAWATO, M., "Human cerebellar activity reflecting an acquired internal model of a new tool," *Nature*, vol. 403, no. 6766, pp. 192–195, 2000.

[67] KAELBLING, L. P., OATES, T., HERNANDEZ, N., and FINNEY, S., "Learning in worlds with objects," in *Working Notes of the AAAI Stanford Spring Symposium on Learning Grounded Representations*, pp. 31–36, 2001.

[68] KAELBLING, L., LITTMAN, M., and MOORE, A., "Reinforcement learning: A survey," *Journal of Artificial Intelligence*, vol. 4, no. 1, pp. 237–285, 1996.

[69] KAJITA, S., KANEHIRO, F., KANEKO, K., FUJIWARA, K., HARADA, K., YOKOI, K., and HIRUKAWA, H., "Biped walking pattern generation by using preview control of zero-moment point," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1620–1626, Citeseer, 2003.

[70] KAKADIARIS, L. and METAXAS, D., "Model-based estimation of 3d human motion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 12, 2000.

[71] KARAMAN, S. and FRAZZOLI, E., "Incremental sampling-based algorithms for optimal motion planning," *arXiv preprint arXiv:1005.0416*, 2010.

[72] KATZ, D., HORRELL, E., YANG, Y., BURNS, B., BUCKLEY, T., GRISHKAN, A., ZHYLKOVSKYY, V., BROCK, O., and LEARNED-MILLER, E., "The UMass mobile manipulator uMan: An experimental platform for autonomous mobile manipulation," in *Workshop on Manipulation in Human Environments at Robotics: Science and Systems*, Citeseer, 2006.

[73] KATZ, D., PYURO, Y., and BROCK, O., "Learning to manipulate articulated objects in unstructured environments using a grounded relational representation," in *Robotics: Science and Systems*, Citeseer, 2008.

[74] KAVRAKI, L. and LATOMBE, J., "Probabilistic roadmaps for robot path planning," *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, vol. 53, 1998.

[75] 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.

[76] KEARNS, M., MANSOUR, Y., and NG, A., "A sparse sampling algorithm for near-optimal planning in large markov decision processes," in *International Joint Conference on Artificial Intelligence*, vol. 16, LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.

[77] KOBER, J. and PETERS, J. R., "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 21* (KOLLER, D., SCHUURMANS, D., BENGIO, Y., and BOTTOU, L., eds.), pp. 849–856, Curran Associates, Inc., 2009.

[78] KOCSIS, L. and SZEPESVÁRI, C., "Bandit based monte-carlo planning," *Machine Learning: ECML 2006*, pp. 282–293, 2006.

[79] KOENIG, N. and HOWARD, A., "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robotics and Systems*, pp. 2149–2154, 2004.

[80] KOLLER, D. and FRIEDMAN, N., *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[81] KUFFNER JR, J. and LAVALLE, S., "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 995–1001, IEEE, 2000.

[82] KULKARNI, T. D., KOHLI, P., TENENBAUM, J. B., and MANSINGHKA, V. K., "Picture: a probabilistic programming language for scene perception," 2015.

[83] KUWATA, Y., FIORE, G. A., TEO, J., FRAZZOLI, E., and J., H. P., "Motion Planning for Urban Driving using RRT," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1620–1626, Citeseer, 2003.

[84] LAMIRAUX, F., BONNAFOUS, D., and LEFEBVRE, O., "Reactive path deformation for nonholonomic mobile robots," *IEEE Transactions on Robotics*, vol. 6, no. 20, pp. 967–977, 2004.

[85] LAMIRAUX, F., BONNAFOUS, D., and LEFEBVRE, O., "Reactive path deformation for nonholonomic mobile robots," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 967–977, 2004.

[86] LANDAU, I. D., *Adaptive control: algorithms, analysis and applications*. Springer, 2011.

[87] LAVALLE, S., "Rapidly-exploring random trees a new tool for path planning," 1998.

[88] LAVALLE, S., *Planning algorithms*. Cambridge university press, 2006.

[89] LAVALLE, S. and KUFFNER, J., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*, p. 293, AK Peters, Ltd., 2001.

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

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

[92] LEFFLER, B. R., LITTMAN, M. L., and EDMUNDS, T., "Efficient reinforcement learning with relocatable action models," in *Proceedings of AAAI*, vol. 22, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[93] LEVIHN, M., KAELBLING, L., LOZANO-PEREZ, T., and STILMAN, M., "Foresight and reconsideration in hierarchical planning and execution," in *IROS*, 2013.

[94] LEVIHN, M., SCHOLZ, J., and STILMAN, M., "Hierarchical decision theoretic planning for navigation among movable obstacles," in *WAFR*, 2012.

[95] LEVIHN, M., SCHOLZ, J., and STILMAN, M., "Planning with movable obstacles in continuous environments with uncertain dynamics," in *ICRA*, May 2013.

[96] LEVIHN, M. and STILMAN, M., "Using environment objects as tools: Unconventional door opening," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2502–2508, IEEE, 2014.

[97] LIKHACHEV, M. and FERGUSON, D., "Planning long dynamically-feasible maneuvers for autonomous vehicles," *International Journal of Robotics Research (IJRR)*, 2009.

[98] LIKHACHEV, M., GORDON, G., and THRUN, S., "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems (NIPS) 16*, Cambridge, MA: MIT Press, 2003.

[99] LIN, Z., ZHANG, W., and TANG, X., "Learning partial differential equations for computer vision," *Peking University the Chinese University of Hong Kong*, 2008.

[100] LIU, C., HERTZMANN, A., and POPOVIĆ, Z., "Learning physics-based motion style with nonlinear inverse optimization," in *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 2005.

[101] LIU, K., "Designing motion guides for ergonomic collaborative manipulation," in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2709–2715, Citeseer, 2000.

[102] LOZANO-PÉREZ, 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.

[103] LUO, J., ZHANG, Y., HAUSER, K., PARK, H. A., PALDHE, M., LEE, C., GREY, M., STILMAN, M., OH, J. H., LEE, J., and OTHERS, "Robust ladder-climbing with a humanoid robot with application to the darpa robotics challenge," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2792–2798, IEEE, 2014.

[104] LYNCH, K., LIU, C., SORENSEN, A., KIM, S., PESHKIN, M., COLGATE, J., TICKEL, T., HANNON, D., and SHILES, K., "Motion guides for assisted manipulation," *The International Journal of Robotics Research*, vol. 21, no. 1, p. 27, 2002.

[105] M. STILMAN, J. K., "Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments," *The International Journal of Humanoid Robotics*, vol. 2, no. 4, pp. 479–504, 2005.

[106] MARDER-EPPSTEIN, E., BERGER, E., FOOTE, T., GERKEY, B., and KONOLIGE, K., "The office marathon: Robust navigation in an indoor office environment," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 300–307, IEEE, 2010.

[107] MARTN, R. M. and BROCK, O., "Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[108] MELCHIOR, N. A. and SIMMONS, R., "Particle rrt for path planning with uncertainty," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1617–1624, IEEE, 2007.

[109] MERICLI, T. A., VELOSO, M., and AKIN, H. L., "Achievable push-manipulation for complex passive mobile objects using past experience," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 71–78, International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[110] MIDDLETONE, R. and GOODWIN, G., "Adaptive computed torque control for rigid link manipulators," in *Decision and Control, 1986 25th IEEE Conference on*, vol. 25, IEEE, 1986.

[111] MÖSENLECHNER, L. and BEETZ, M., "Using physics-and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior.," in *ICAPS*, 2009.

[112] MUELLING, K., KOBER, J., and PETERS, J., "Learning table tennis with a mixture of motor primitives," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, IEEE, 2010.

[113] MURRAY, R. and SASTRY, S., *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[114] MUSSA-IVALDI, F., "Modular features of motor control and learning," *Current opinion in neurobiology*, vol. 9, no. 6, pp. 713–717, 1999.

[115] NEAL, R., "Markov chain sampling methods for dirichlet process mixture models," *Journal of computational and graphical statistics*, vol. 9, no. 2, 2000.

[116] NEAL, R. M., "Mcmc using hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, 2011.

[117] NG, A., COATES, A., DIEL, M., GANAPATHI, V., SCHULTE, J., TSE, B., BERGER, E., and LIANG, E., "Autonomous inverted helicopter flight via reinforcement learning," *Experimental Robotics IX*, 2006.

[118] NGUYEN-TUONG, D. and PETERS, J., "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, 2011.

[119] NIEBERGALL, M. and HAHN, H., "Identification of the ten inertia parameters of a rigid body," *Nonlinear Dynamics*, vol. 13, no. 4, 1997.

[120] NOZAWA, S., MAKI, T., KOJIMA, M., KANZAKI, S., OKADA, K., and INABA, M., "Wheelchair Support by a Humanoid Through Integrating Environment Recognition, Whole-body Control and Human-Interface Behind the User," 2008.

[121] PARK, F. C. and HAAN, J., "srlib - snu robot dynamics library." http://r-station.co.kr/srlib/.

[122] PATIL, A., HUARD, D., and FONNESBECK, C. J., "Pymc: Bayesian stochastic modelling in python," *Journal of statistical software*, vol. 35, no. 4, p. 1, 2010.

[123] PETERS, J. and SCHAAL, S., "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, 2008.

[124] PETERS, J., VIJAYAKUMAR, S., and SCHAAL, S., "Reinforcement learning for humanoid robotics," in *Proc. 3rd IEEE-RAS Intl Conf. on Humanoid Robots*, pp. 29–30, Citeseer, 2003.

[125] PIVTORAIKO, M. and KELLY, A., "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 3231–3237, 2005.

[126] POUPART, P., VLASSIS, N., HOEY, J., and REGAN, K., "An analytic solution to discrete bayesian reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006.

[127] PRATT, G. and MANZO, J., "The darpa robotics challenge [competitions]," *Robotics & Automation Magazine, IEEE*, vol. 20, no. 2, pp. 10–12, 2013.

[128] PUNJANI, A. and ABBEEL, P., "Deep learning helicopter dynamics models," 2015.

[129] QUIGLEY, M., GERKEY, B., CONLEY, K., FAUST, J., FOOTE, T., LEIBS, J., BERGER, E., WHEELER, R., and NG, A., "ROS: an open-source Robot Operating System," in *International Conference on Robotics and Automation*, 2009.

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

[131] RASMUSSEN, C., KUSS, M., and OTHERS, "Gaussian processes in reinforcement learning," *Advances in neural information processing systems*, vol. 16, 2004.

[132] RINGERT, J. O., RUMPE, B., and WORTMANN, A., "A case study on model-based development of robotic systems using montiarc with embedded automata," *arXiv preprint arXiv:1408.5692*, 2014.

[133] RUS, D., "Coordinated manipulation of objects in a plane," *Algorithmica*, vol. 19, no. 1, pp. 129–147, 1997.

[134] RUSSELL, S., "Artificial intelligence: A modern approach author: Stuart russell, peter norvig, publisher: Prentice hall pa," 2009.

[135] RUSU, R. B., ŞUCAN, I. A., GERKEY, B., CHITTA, S., BEETZ, M., and KAVRAKI, L. E., "Real-time perception guided motion planning for a personal robot," (St. Louis, USA), October 2009.

[136] SAFONOVA, A., POLLARD, N., and HODGINS, J., "Optimizing human motion for the control of a humanoid robot," in *2nd International Symposium on Adaptive Motion of Animals and Machines (AMAM2003)*, Citeseer, 2003.

[137] SANBORN, A. N., MANSINGHKA, V. K., and GRIFFITHS, T. L., "Reconciling intuitive physics and newtonian mechanics for colliding objects.," *Psychological review*, vol. 120, no. 2, p. 411, 2013.

[138] SARAMAGO, S. and STEFFEN, V., "Optimization of the trajectory planning of robot manipulators taking into account the dynamics of the system," *Mechanism and machine theory*, vol. 33, no. 7, pp. 883–894, 1998.

[139] SCHAAL, S. and ATKESON, C., "Robot juggling: implementation of memory-based learning," *Control Systems, IEEE*, vol. 14, no. 1, 1994.

[140] SCHAAL, S., ATKESON, C., and VIJAYAKUMAR, S., "Real-time robot learning with locally weighted statistical learning," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, vol. 1, IEEE, 2000.

[141] SCHOLZ, J., LEVIHN, M., ISBELL, C., and WINGATE, D., "A physics-based model prior for object-oriented mdps," 2014.

[142] SCHOLZ, J., CHITTA, S., MARTHI, B., and LIKHACHEV, M., "Cart pushing with a mobile manipulation system: Towards navigation with moveable objects," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 6115–6120, IEEE, 2011.

[143] SCHOLZ, J., LEVIHN, M., ISBELL, C. L., CHRISTENSEN, H., and STILMAN, M., "Learning non-holonomic object models for mobile manipulation," 2015.

[144] SCHOLZ, J., LEVIHN, M., ISBELL, C. L., and WINGATE, D., "A physics-based model prior for object-oriented mdps," in *International Conference on Machine Learning*, 2014.

[145] SCHOLZ, J. and STILMAN, M., "Combining motion planning and optimization for flexible robot manipulation," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, IEEE, 2010.

[146] SCIAVICCO, L. and SICILIANO, B., *Modelling and control of robot manipulators*. Springer Verlag, 2000.

[147] SENTIS, L. and KHATIB, O., "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *International Journal of Humanoid Robotics*, vol. 2, no. 4, pp. 505–518, 2005.

[148] SHIGEMI, S., KAWAGUCHI, Y., YOSHIIKE, T., KAWABE, K., and OGAWA, N., "Development of New ASIMO," *HONDA R AND D TECHNICAL REVIEW*, vol. 18, no. 1, p. 38, 2006.

[149] SHKOLNIK, A., LEVASHOV, M., ITANI, S., and TEDRAKE, R., "Motion planning for bounding on rough terrain with the littledog robot," in *Submitted to the International Conference on Robotics and Automation, Anchorage, Alaska. IEEE/RAS*, 2010.

[150] SICILIANO, B. and VILLANI, L., *Robot force control*, vol. 540. Kluwer Academic Pub, 1999.

[151] SIMEON, T., LAUMOND, J., CORTES, J., and SAHBANI, A., "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, p. 729, 2004.

[152] SONTAG, E., *Mathematical control theory: deterministic finite dimensional systems*, vol. 6. Springer, 1998.

[153] SPELKE, E., "Initial knowledge: Six suggestions," *Cognition*, vol. 50, no. 1, 1994.

[154] SPELKE, E. S., "Principles of object perception," *Cognitive science*, vol. 14, no. 1, pp. 29–56, 1990.

[155] STAM, J., "Stable fluids," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 1999.

[156] STENTZ, A. and HEBERT, M., "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, pp. 127–145, 1995.

[157] STILMAN, M., "Task constrained motion planning in robot joint space," *IROS07*, 2007.

[158] STILMAN, M. and KUFFNER, J., "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Proc. IEEE Int. Conf. on Humanoid Robotics (Humanoids'04)*, 2004.

[159] STILMAN, M., NISHIWAKI, K., and KAGAMI, S., "Learning object models for whole body manipulation," in *Int. Conf. on Humanoid Robotics (Humanoids 07)*, Citeseer, 2007.

[160] STILMAN, M., NISHIWAKI, K., and KAGAMI, S., "Humanoid teleoperation for whole body manipulation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008.

[161] STILMAN, M., NISHIWAKI, K., KAGAMI, S., and KUFFNER, J., "Planning and executing navigation among movable obstacles," in *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, pp. 820 – 826, October 2006.

[162] STILMAN, M., SCHAMBUREK, J., KUFFNER, J., and ASFOUR, T., "Manipulation planning among movable obstacles," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3327–3332, Citeseer, 2007.

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

[164] STILMAN, M., OLSON, J., and GLOSS, W., "Golem krang: Dynamically stable humanoid robot for mobile manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 3304–3309, May 2010.

[165] SUTTON, R. S. and BARTO, A. G., *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[166] SUTTON, R. S., MCALLESTER, D. A., SINGH, S. P., MANSOUR, Y., and OTHERS, "Policy gradient methods for reinforcement learning with function approximation.," in *NIPS*, vol. 99, pp. 1057–1063, Citeseer, 1999.

[167] SUTTON, R. and BARTO, A., *Reinforcement Learning: An Introduction*. MIT Press, 2004.

[168] TALTON, J., YANG, L., KUMAR, R., LIM, M., GOODMAN, N., and MĚCH, R., "Learning design patterns with bayesian grammar induction," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 63–74, ACM, 2012.

[169] TAN, J. and XI, N., "Integrated sensing and control of mobile manipulators," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2, pp. 865–870, 2001.

[170] TAN, J. and XI, N., "Unified model approach for planning and control of mobile manipulators," in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 3145–3152, IEEE; 1999, 2001.

[171] TAN, J., XI, N., and WANG, Y., "Integrated task planning and control for mobile manipulators," *The International Journal of Robotics Research*, vol. 22, no. 5, p. 337, 2003.

[172] TEDRAKE, R., ZHANG, T., and SEUNG, H., "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, IEEE, 2004.

[173] THEODOROU, E., BUCHLI, J., and SCHAAL, S., "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2397–2403, IEEE, 2010.

[174] THOMAZ, A. and BREAZEAL, C., "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, p. 1000, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[175] TODOROV, E., "Optimality principles in sensorimotor control," *Nature Neuroscience*, vol. 7, no. 9, pp. 907–915, 2004.

[176] TODOROV, E., "A convex, smooth and invertible contact model for trajectory optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1071–1076, IEEE, 2011.

[177] 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 (WAFR VIII)*, pp. 599–614, 2008.

[178] VLASSIS, N., GHAVAMZADEH, M., MANNOR, S., and POUPART, P., "Bayesian reinforcement learning," in *Reinforcement Learning*, Springer, 2012.

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

[180] WANG, T., LIZOTTE, D., BOWLING, M., and SCHUURMANS, D., "Bayesian sparse sampling for on-line reward optimization," in *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005.

[181] WILFONG, G., "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*, ACM, 1988.

[182] WILLIAMS, R. J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[183] WITKIN, A., "Physically based modeling: Principles and practice constrained dynamics," *SIGGRAPH Course notes*, pp. 11–21, 1997.

[184] WOLPERT, D. M., DOYA, K., and KAWATO, M., "A unifying computational framework for motor control and social interaction," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 358, no. 1431, pp. 593–602, 2003.

[185] WOLPERT, D. M. and KAWATO, M., "Multiple paired forward and inverse models for motor control," *Neural networks*, vol. 11, no. 7, pp. 1317–1329, 1998.

[186] WOLPERT, D. and GHAHRAMANI, Z., "Computational principles of movement neuroscience," *nature neuroscience*, vol. 3, pp. 1212–1217, 2000.

[187] WU, H., LEVIHN, M., and STILMAN, M., "Navigation among movable obstacles in unknown environments," in *IROS*, October 2010.

[188] YAMAMOTO, Y. and YUN, X., "Coordinating locomotion and manipulation of a mobile manipulator," *IEEE Transactions on Automatic Control*, vol. 39, no. 6, pp. 1326–1332, 1994.

[189] ZHU, C., BYRD, R. H., LU, P., and NOCEDAL, J., "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.