Navigation Among Movable Obstacles

Mike Stilman

CMU-RI-TR-07-37

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
October, 2007

Thesis Committee:

James J. Kuffner
Christopher G. Atkeson
Matthew T. Mason
Jean-Claude Latombe

ABSTRACT

Navigation Among Movable Obstacles

Mike Stilman

Robots would be much more useful if they could move obstacles out of the way. Traditional motion planning searches for collision free paths from a start to a goal. However, real world search and rescue, construction, home and nursing home domains contain debris, materials clutter, doors and objects that need to be moved by the robot. Theoretically, one can represent all possible interactions between the robot and movable objects as a huge search. We present methods that simplify the problem and make Navigation Among Movable Obstacles (NAMO) a practical challenge that can be addressed with current computers.

This thesis gives a full development cycle from motion planning to implementation on a humanoid robot. First, we devise a state space decomposition strategy that reasons about free space connectivity to select objects and identify helpful displacements. Second, we present controls for balance and manipulation that allow the robot to move objects with previously unknown dynamics. Finally, we combine these results in a complete system that recognizes environment objects and executes Navigation Among Movable Obstacles.

Our continued work in NAMO planning has focused on three topics: reasoning about object interaction, three dimensional manipulation and interaction with constrained objects. This thesis presents the computational and theoretical challenges that arise from these elaborations of the NAMO domain. In each case we introduce extensions to our algorithms that respond to the challenge and evaluate their performance in simulation. All the methods presented in this thesis not only solve previously unsolved problems but also operate efficiently, giving real-time results that can be used during online operation.

Acknowledgments

I am grateful to my advisors, James Kuffner and Chris Atkeson for their support and guidance throughout grad school. James introduced me to motion planning, opened wonderful opportunities for collaboration and gave me the confidence to pursue NAMO. Chris inspired me to see the bigger picture of intelligent robotics and guided me through many difficult times with unwavering integrity and wisdom.

Huge thanks to my collaborators, Koichi Nishiwaki and Satoshi Kagami for countless contributions and many fun, sleepless nights of coding and experiments! I am grateful to Jan-Ullrich Schamburek for developing Chapter 7 and getting me to relive the excitement of learning. I thank Matt Mason for his many creative thoughts on manipulation and his wonderful company at conferences. Jean-Claude Latombe's thorough insight motivated me to sharpen my understanding and grow as a researcher.

I am forever grateful to all the Stanford folks. Oussama Khatib inspired me to pursue robotics and welcomed me into his lab. John McCarthy got me excited about math and taught me to "write it up." I especially thank Jaeheung Park and Eugene Davydov for their many years of teaching, advice and most importantly friendship! Thank you Eric Curiel for existing and Trevor Blackwell for helping me believe in dreams.

Adam Sanphy and Young-Woo Seo have been wonderful friends. Thank you for your sincerity, kindness and for taking me seriously on all my crazy plans. I am grateful to Coach D, Sir John, Matt, Reese and all the guys at boxing for giving me something better to do with my head than worry.

I thank Sajid Siddiqi, Marius Leordeanu and Marliese Bonk for keeping me company in the best of times and keeping me sane in the worst. Joel Chestnutt, Phil Michel, Ross Knepper and Tom Howard, it was great hanging out with you around the world! Thank you to all the lab members, RI people and Humanoids folks. Martin Stolle, thank you for being the best officemate ever.

Al Rizzi, Steve LaValle, Andrew Moore, Reid Simmons and Martial Hebert gave me incredibly rich perspectives on robotics. Thank you for the chats, classes and seminars! Thank you to Garth Zeglin, Darrin Bentivegna and Clark Haynes for helping me survive through grad school. Suzanne Lyons Muth, thank you for everything!

I thank my parents for raising me to be a scientist and engineer. My dad, Boris got me excited about computers and AI and my mom, Zina showed me what it means to be a great teacher. I am grateful to my grandparents for their love and support. Thank you to Minoru and Keiko Tanaka for accepting me and welcoming me into their home.

Finally, I thank the beautiful, crazy girl that turned my world upside down. Akiko, my wife, inspired, listened, loved, cared and made our life together into a fantastic adventure. Of all the luck in my life - she tops the list!

Contents

1 Introduction **1**

1.1 Motivation for NAMO 1

1.2 Related Work 3

1.3 Challenges 5

1.4 Overview 7

2 NAMO Planning Domain **8**

2.1 Problem Statement 8

2.2 Operators 9

2.3 Action Spaces 10

2.4 Complexity of Search 12

3 NAMO Planning **15**

3.1 Overview 15

3.2 Configuration Space 15

3.3 Goals for Navigation 17

3.4 Goals for Manipulation 19

3.5 Planning as Graph Search 20

3.5.1 Linear Problems 21

3.5.2 Local Manipulation Search 22

3.5.3 Connecting Free Space 23

3.5.4 Analysis 24

3.5.5 Challenges of ConnectFS 26

3.6 Planner Prototype 27

3.6.1 Relaxed Constraint Heuristic 27

3.6.2 High Level Planner . . . 30
3.6.3 Examples and Experimental Results . . . 30
3.6.4 Analysis . . . 33
3.7 Summary . . . 35

4 Humanoid Manipulation **37**
4.1 Related Work . . . 38
4.2 Biped Control with External Forces . . . 39
4.2.1 Decoupled Positioning . . . 40
4.2.2 Trajectory Generation . . . 41
4.2.3 Online Feedback . . . 43
4.3 Modeling Object Dynamics . . . 43
4.3.1 Motivation for Learning Models . . . 44
4.3.2 Modeling Method . . . 44
4.4 Experiments and Results . . . 45
4.4.1 Prediction Accuracy . . . 46
4.4.2 System Stability . . . 47
4.4.3 Discussion . . . 48
4.5 Summary . . . 49

5 System Integration **51**
5.1 From Planning to Execution . . . 51
5.2 Measurement . . . 52
5.2.1 Object Mesh Modeling . . . 53
5.2.2 Recognition and Localization . . . 54
5.3 Planning . . . 55
5.3.1 Configuration Space . . . 56
5.3.2 Contact Selection . . . 57
5.3.3 Action Spaces . . . 58
5.4 Uncertainty . . . 59
5.4.1 Impedance Control . . . 60
5.4.2 Replanning Walking Paths . . . 60
5.4.3 Guarded Grasping . . . 61
5.5 Results . . . 62

5.6 Summary 62

6 Object Interaction **64**

6.1 Introduction 64

6.2 Extended Hierarchy 66

6.3 Planning in Monotone Domains 67

6.3.1 Forward Search 67

6.3.2 Reverse Search 68

6.4 Artificial Constraints 69

6.5 Algorithm 70

6.5.1 Obstacle Identification 71

6.5.2 Constraint Resolution 72

6.5.3 Depth First Search 75

6.6 Implementation 77

6.6.1 Planning Details 77

6.6.2 Results 78

6.6.3 Complexity 78

6.7 Summary 79

7 3D Movable Obstacles **81**

7.1 Broader Applications 81

7.2 Domain Observations 82

7.2.1 Simplifying Assumptions 83

7.2.2 Challenges 83

7.3 Algorithm 85

7.4 Motion Sampling 87

7.4.1 Sampling Placements 87

7.4.2 Sampling Paths 89

7.5 Constraints 92

7.5.1 Placement Constraints 92

7.5.2 Motion Constraints 93

7.6 Results 94

7.7 Summary 96

8 Constrained Objects **97**

8.1 Introduction . . . 97

8.2 Related Work . . . 99

8.3 Preliminaries . . . 100

8.4 Specifying Constraints . . . 101

8.5 Introducing Constrained Sampling . . . 104

8.5.1 Computing Distance . . . 104

8.5.2 Baseline: Randomized Gradient Descent . . . 105

8.6 Relating Joint Motion to Constraint Error . . . 106

8.6.1 Tangent Space Sampling . . . 108

8.6.2 First-Order Retraction . . . 109

8.7 Results . . . 110

8.8 Extensions . . . 113

8.9 Summary . . . 115

9 Conclusion **117**

A Dynamic $\mathcal{C}$-space Modification **118**

B Preview Control **120**

B.1 Transformation to LQI . . . 121

B.2 Biped Application . . . 123

C Coordinate Transformations **124**

List of Figures

1.1 Planned solution to a NAMO problem. The robot H7 repositions a chair to make space for a navigation path. 2

2.1 Simple NAMO problems on a planar grid. 13

3.1 Simulated 2D NAMO $\mathcal{C}_R$-space for a circular robot. 18
3.2 NAMO $\mathcal{C}_R$-space partitioned into components. 19
3.3 Construction of FCT structures the solution for L_1 problems. 24
3.4 Pseudo-code for RCH. 28
3.5 Pseudo-code for SelectConnect. 30
3.6 Walk-through of an autonomous SelectConnect. 31
3.7 The generated plan output by our dynamic simulation NAMO planner is illustrated by the time-lapse sequences on the right. 32
3.8 A larger scale example consisting of 90 movable obstacles. Two separate plans are computed and demonstrated in our dynamic simulation. 33
3.9 SelectConnect solves the open problem considered difficult by Chen. . 36

4.1 Model of the robot and object used in our work. 41
4.2 HRP-2 pushes then pulls 30 and 55kg loaded tables. 46
4.3 Comparison of forces experienced with reactive and model-based control. . 48
4.4 Trajectory error introduced by preview control with erroneous prediction. 49
4.5 Realized ZMP for identical reference trajectories 49
4.6 Manipulation of a 55kg table with 30kg model. 50

5.1 Autonomous execution of NAMO with architecture diagram. 52
5.2 Off-line modeling and online localization of a chair. 54
5.3 Object localization procedure. 55
5.4 Mapping objects into the planning domain. 57

5.5 Simulated example shows NAMO plan and traces of execution. 59

6.1 Simulated solution requires the robot to move four objects. 65
6.2 Pseudo-code for IDENTIFY-OBSTACLE. 71
6.3 RCH_{last} selects subgoals for constraint resolution. 71
6.4 Pseudo-code for RESOLVE-CONSTRAINTS called by IDENTIFY-OBSTACLE. 75
6.5 RESOLVE-CONSTRAINTS displaces table to make space for moving the couch. 75
6.6 A search tree for the given example. Large upward arrows indicate backtracking when an object cannot be resolved. 76

7.1 Mobile manipulator moves the gear and fan before retrieving the hammer. 82
7.2 Simulated workspace and configuration space 84
7.3 RSC Algorithm pseudo-code and diagram. 86
7.4 Autonomously selected placements for the drill. 88
7.5 Basic RRT Algorithm Pseudo Code 90
7.6 Multi-Goal RRT-Connect pseudo code. 91
7.7 Simulated 8-DOF manipulator delivers an obstructed drill. 94
7.8 Simulated planar arm retrieves a green block from the cubicle. 95

8.1 DOOR: Computed solution to door opening avoids obstacles and joint limits. 98
8.2 DOOR: Paths for robot joints and door rotation about the z-axis. 98
8.3 Roll/pitch/yaw constraints. (c-d) are parameterized by robot configuration. 103
8.4 Pseudo-code for Task-Constrained RRT (TC-RRT) construction. 106
8.5 Pseudo-code for Randomized Gradient Descent sampling. 108
8.6 Pseudo-code for Tangent Space Sampling 108
8.7 FR_NEW_CONFIG Algorithm pseudo-code and diagram. 110
8.8 DRAWER: Solved drawer opening with obstacle avoidance. 111
8.9 PAINT: Simulation requiring upright transport of paint. 112
8.10 PAINT (FR): Paths for joints 2-8 in Paint experiment. 112

A.1 Update of dynamic $\mathcal{C}_R$ by reference counting. 119

B.1 Computed torso trajectory for five steps. 121

List of Tables

4.1 Prediction Accuracy 47
4.2 System Stability 47

5.1 NAMO Implementation: Run times for Planning and Execution 62

6.1 Quantities of objects and running times for examples in figures. 78

7.1 RSC: Run times for algorithm components 95
7.2 Average cost for each procedure over all experiments. 95

8.1 Runtimes for Experiment 1: Door 113
8.2 Runtimes for Experiment 2: Drawer 113
8.3 Runtimes for Experiment 3: Paint 114

1 Introduction

1.1 Motivation for NAMO

Robots would be much more useful if they could move obstacles out of the way. In this thesis, we introduce algorithms for autonomous Navigation Among Movable Obstacles (NAMO), evaluate their effectiveness in simulation and present a successful implementation of NAMO on a humanoid robot. Traditional motion planning searches for collision free paths from a start to a goal. However, real world domains like search and rescue, construction, home robots and nursing home assistance contain debris, materials clutter, doors and objects that need to be moved by the robot. Theoretically, one can represent all possible interactions between the robot and movable objects as a huge search. We will present two methods that use state space decomposition and heuristic search to simplify the problem and find practical solutions in common environments.

The ability to solve NAMO problems has direct applications in domains that require robot autonomy for safety and assistance. Consider a search and rescue mission to extract the victims of a natural disaster such as Hurricane Katrina. Already robots enter these environments to minimize the exposure of humans to unstable structures, gases and other hazards. However, collapsed rubble and objects displaced by floodwaters cause previously navigable passages to become blocked. In solving NAMO, the robot examines its environ-

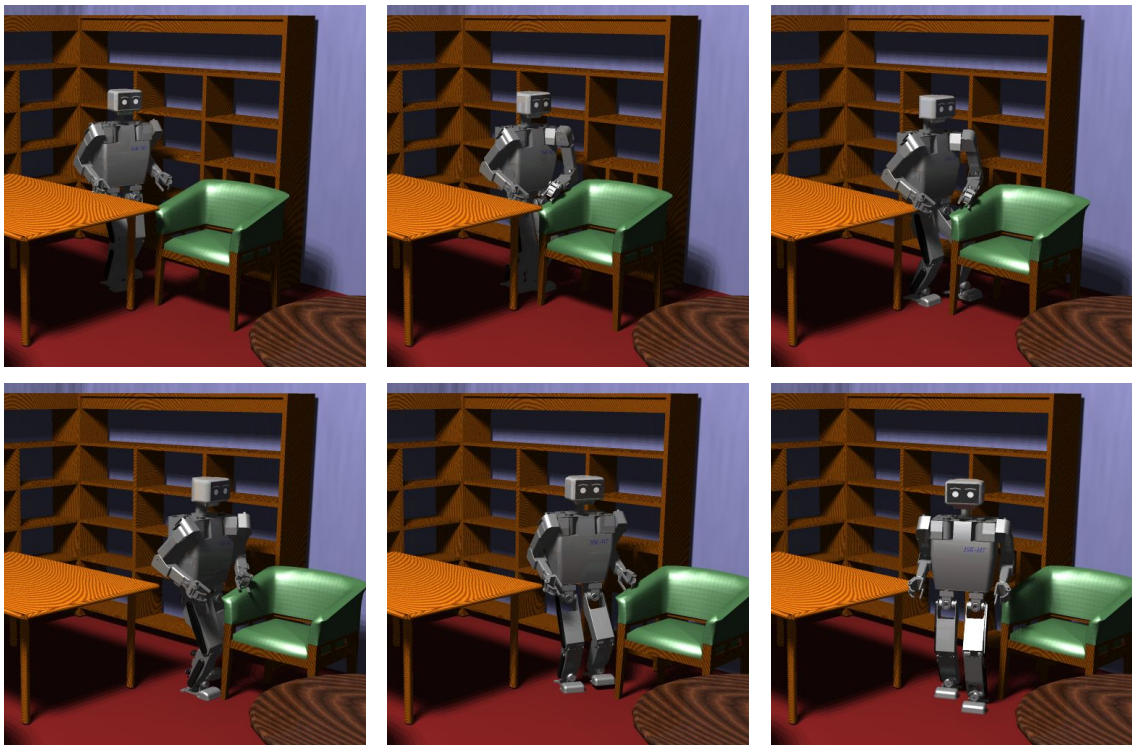


Figure 1.1: Planned solution to a NAMO problem. The robot H7 repositions a chair to make space for a navigation path.

ment, decides which objects must be moved and clears a way to further progress. Human safety is one of many motivations for NAMO. Consider the comfort and self-reliance of the elderly and disabled. In 2006, Nursing Economic reported that the United States shortage of nurses would grow to eight times the current size by the year 2020 [1]. Robots that help patients get around, reach for medicine and food alleviate this need and provide independence as well as personal autonomy. Yet, in contrast to laboratories and factories, human domains have movable objects that are often misplaced. In Figure 1.1, even a chair in front of a table challenges the robot to Navigate Among Movable Obstacles.

Robots that have the physical capacity for removing chairs and rubble are known as mobile manipulators. Some examples include the Kawada HRP-2 [2], Stanford SAMM [3], Remotec Field Robots [4] and the Vecna Bear. Manus robot arms have recently been attached to wheelchairs for assisting the disabled [5]. Further development in sensing, planning and control will increase the utility of such robots, making them more autonomous

and effective at handling the real world. We believe that Navigation Among Movable Obstacles will be a primitive skill for mobile manipulators and therefore a fundamental challenge for research.

1.2 Related Work

In contrast to planning and scheduling with general axioms this thesis focuses on a form of planning that we call *spatial reasoning*. The robot is required to think about the geometry and topology of the space that it occupies, the constraints it faces and how these constraints can be modified to allow the robot to perform its task. In NAMO, the constraints are geometric obstacles that interfere with navigation.

To our knowledge, the first example of an reasoning about constraints was addressed by the SRI robot Shakey [6]. Among other axioms, the robot represented the locations of boxes in predicate calculus. Specialized predicates indicated states in which a box blocked entry to a door. When planning to move through the doorway, the robot would first plan to push the box to one of two locations, thereby negating the "blocked" predicate. This autonomous decision was more closely related to door opening than NAMO since boxes that block arbitrary navigation paths were not considered for motion.

The work at SRI was succeeded by MIT investigations of assembly planning. [7, 8, 9, 10] studied methods for composing a number of parts into a target assembly. Although this research did not allow interaction with the environment, the required temporal/spatial coordination of part positioning yielded a number of interesting and relevant results. For instance, the FINDSPACE problem of establishing a volume where a specified geometric object would fit could be used to locate manipulation subgoals for any planner [9]. In the context of dynamics, Inoue's peg insertion algorithm formed a basis for employing force control to gain increased resolution in fine manipulation [11]. Perhaps the most relevant result to our work can be found in [12], defininining pre-image backchaining as a method for constructing solutions based on reasoning about spatial constraints.

Further research in assembly planning encouraged a dichotomy between reasoning

about multiple objects and precise planning for a single object. Typically, assembly planners have diverged from the primary interests of our domain. These planners focus on separating a collection of parts and ignore the robot/manipulator. Planning actions allow unassembled parts to be removed to "infinity."[13, 14] Once parts are removed they no longer affect future manipulation. In planning among movable obstacles, both the robot and objects are constrained by environment geometry and the robot's kinematics. These constraints limit the robot's ability to grasp and manipulate objects. Most importantly, they require the planner to consider the effects of displacing objects on the robot's ability to navigate and manipulate in the future.

In contrast, *manipulation planning* carefully studies the interaction between a robot and a single object. For instance, Lynch developed bounds for stable pushing motions by a mobile robot with edge-edge contact between the robot and a specified object [15, 16]. The proposed search algorithm branched on possible motions within these bounds and yield stable plans for repositioning the object among obstacles. Alami et.al. pursued a similar problem with rigid grasping by constructing graphs of transit(navigation) and transfer(manipulation) paths for robot/object motion [17]. This work was later extended by Simeon et.al. to handle continuous grasps and placements [18]. Erdmann and Kaelbling presented different strategies for handling sensing and action uncertainty during planning to guarantee or optimize the validity of plans [19, 20].

The void between multi-object assembly and fine manipulation planning for a single object is addressed by a less studied but highly relevant extension to assembly planning known as the *rearrangement problem*.[21] Rearrangement can be viewed as any real-world generalization of the Sokoban puzzle: a robot is required to move a set of objects between specified initial and final configurations.[22] Although the task of the robot is akin to the earlier mentioned manipulation work, the key difference is that it may not be possible to directly achieve the individual object placement goals. Rearrangement planners typically construct precedence graphs and determine intermediate configurations for obstacles.[23] Since intermediate configurations are not specified in the problem statement, the curse

of dimensionality in this problem is very similar to NAMO. The most experimentally successful rearrangement planners employ opportunistic methods in selecting object perturbations that allow for simplified planning.[21, 24]

Almost all the mentioned developments in navigation, manipulation and assembly address problems with fully specified final configurations. To our knowledge, the only planner that handled many movable objects prior to our work was developed by Chen et. al.[25] The authors allowed the robot to "*shove_aside*" and "*push_forward*" obstacles while planning navigation to the goal. We summarize this planner as a search for a *path of least resistance* (PLR). A global planner heuristically evaluates the cost of moving obstacles away from randomly selected points in the domain. The algorithm then searches a graph of neighboring points to form a trajectory of least cost. This trajectory is verified by a local planner that can apply manipulation primitives to connect two neighboring points. PLR is effective on some examples, yet it has some important drawbacks. For each robot location that is found to be reachable, the algorithm considers only one trajectory that led to that event. The authors recognize this lack of backtracking as a cause for incompleteness and show that even slight improvements to their framework would cause exponential growth in complexity. Furthermore, since manipulation is restricted to validating the connectivity of *neighboring points*, manipulation tasks that affect distant portions of the environment are never considered.

1.3 Challenges

Our goal in this thesis is to develop practical algorithms that allow robots to displace obstacles in realistic environments. The challenges we address come from two sources: NAMO planning in very high dimensional spaces and the implementation of physical interactions between a robot and its environment. The former challenge is related to the size of the space, the lack of meaningful heuristics for choosing objects and the complex nonlinearity of object interaction that leads to local minima. The latter requires us to handle measurement or localization and perform multi-objective control for locomotion,

manipulation and compliance while interacting with large environment objects.

First, consider the complexity of planning. In contrast to planning for a single mobile robot with a fixed number of joints or degrees of freedom we are required to consider the displacement of any movable object. In effect, the size of our search space has the same order of magnitude as that of a robot that has as many joints as there are obstacles in the environment. This observation would not be so daunting if we had a heuristic for deciding which objects should be moved or where to move them. However, existing distance measures to the goal are not satisfactory heuristics. Pushing the box towards the goal in Figure 2.1(b) blocks the robot's path. The correct solution pulls the box away. In some environments it is difficult to even decide which objects should be moved. Figure 3.6.3 requires the planer to move less than six objects in each plan although the environment contains 90 obstacles. In contrast, Figure 6.1 is an example where all four objects must be moved while three of them do not directly block any path to the goal. In order to obtain an efficient solution, the robot will need heuristics that take into account future plans for navigation and manipulation. These are the concepts that motivate our planning algorithms.

Second, observe the problem of extending algorithms that operate in a simplified 2D environment to both planning for articulated manipulators and execution on a humanoid robot platform. The serious challenges of recognition and localization of objects are beyond the scope of this thesis. We present a simplified scheme that proved sufficient for experimentation and potentially suitable for instrumented environments such as nursing homes, hospitals and private residences. Our focus is on the problems that come after the objects have been localized and a high level plan to move them has been constructed. The robot is required to walk while manipulating objects. Consequently it must satisfy the dynamic balance of locomotion exert the necessary forces for manipulation and exhibit compliance such that it does not damage itself or the environment. All these goals must be achieved simultaneously while staying within a limited workspace for both the legs and arms of the humanoid robot.

1.4 Overview

In this thesis, we determine a movement strategy for the robot. Through implementation, we also present initial results in control and perception that realize the desired movement. Chapter 2 introduces NAMO as a robot planning domain and presents its technical challenges. Chapter 3 gives an initial practical algorithm for solving NAMO problems and evaluates its performance. Chapter 4 validates the practicality of our plans with a controller for humanoid walking and whole body manipulation. Chapter 5 presents an architecture for combining planning with perception and control to execute the NAMO task. Chapters 6, 7 and 8 extend NAMO planning to domains with interacting obstacles, 3D manipulation and constrained objects. In each case we describe the challenges and evaluate our solutions.

2

NAMO Planning Domain

2.1 Problem Statement

We begin our analysis of the NAMO domain as an instance of geometric motion planning [26]. During planning, we assume that the geometry and kinematics of the environment and the robot are known. We also assume that there is no uncertainty in sensing and the effects of robot actions. These assumptions are softened during implementation through active modeling and replanning. We represent objects and robot links as polyhedrons. The environment objects are classified as either fixed or movable.

Formally, the environment is modeled as a 2D or 3D Euclidian space that contains the following items:

- $\mathbf{O}_F = \{F_1, \ldots, F_f\}$ - a set of polyhedral *Fixed Obstacles* that must be avoided.
- $\mathbf{O}_M = \{O_1, \ldots, O_m\}$ - a set of *Movable Obstacles* that the robot can manipulate.
- R - a manipulator with n degrees of freedom represented by polyhedral links.

While paths may not be explicitly parameterized by time, we will use the variable t to refer to a chronological ordering on states and operations. At any time t, the world state W^t defines the position and orientation of the robot links and each object. We represent the world state as follows:

$$W^t = (t, r^t, q_1^t, q_2^t, \ldots, q_m^t)$$

Given an initial configuration W^0 of the robot r^0 and each movable obstacle q_i^0, the goal is to achieve a final configuration r^f for the manipulator.

2.2 Operators

In order to achieve the goal configuration the robot is permitted to change its own configuration and possibly the configuration of one grasped obstacle at any time step. Between time steps, any change to the robot joints must be continuous. We therefore interpret any "change" as an action that follows a path or trajectory.

We can distinguish between two primitive operators or actions: *Navigate* and *Manipulate*. Each action is parameterized by a path $\tau(r_i, r_j)$ that defines the motion of the robot between two configurations: $\tau : [0, 1] \rightarrow r$ where $\tau[0] = r_i$ and $\tau[1] = r_j$.

The $Navigate$ operator refers to contact-free motion. While the robot may be in sliding contact with an object, its motion must not displace any objects by collision or friction. $Navigate$ simply moves the robot joints as specified by τ.

$$Navigate : (W^t, \tau(r^t, r^{t+1})) \rightarrow W^{t+1} \tag{2.2.1}$$

When the robot motion affects the environment by displacing an object, O_i, we refer to the action as $Manipulate$. The manipulate operator consists of two paths: one for the robot and one for O_i. Since the object is not autonomous, the object path is parameterized by the robot path and the initial contact or grasp $G_i \in \mathbf{G}(O_i)$. The set $\mathbf{G}(O_i)$ consists of relative transformations between the robot end-effector and the object that constitute contact.

$$Manipulate : (W^t, O_i, G_i, \tau(r^t, r^{t+1})) \rightarrow W^{t+1} \tag{2.2.2}$$

Distinct G_i lead to different object motions given the same end-effector trajectory. We let $\tau_o = ManipPath(G_i, \tau)$ be the interpretation of the path for the object during manipulation according to the constraints imposed by the contact. *Manipulate* maps a world state, contact and path to a new world state where the robot and O_i have been displaced.

The action is valid when neither the robot nor object collide or displace other objects. Validity is also subject to the constraints discussed in Section 2.3.

The two action descriptions point to a general formulation for interacting with environment objects. The robot iterates a two step procedure. First, it moves to a contact state with the *Navigate* operator and then applies a *Manipulate* operator to displace the object. The robot also uses *Navigate* to reach a goal state.

2.3 Action Spaces

In Section 2.2 we left the parameters for *Manipulate* open to interpretation. This is consistent with our focus on deciding a high level movement strategy for the robot in Chapters 2 and 3. In this section we give three classes of parameterizations for manipulating objects. In each case, the class translates the trajectory of the robot into a motion for the object. We point out the relative advantages and disadvantages of the classes with regard to modeling requirements, generality and reliability.

Grasping (Constrained Contact): The simplest method for translating robot motion into object motion can be applied when the object is rigidly grasped by the robot. The *Constrained Contact (***CC***)* interpetation of our actions relates them directly to $Transit$ and $Transfer$ operators described by Alami [17]. A grasped object remains at a fixed transform relative to the robot end-effector. To move an object, the robot must first *Navigate* to a grasping configuration and then *Manipulate*.

In addition to requiring collision-free paths, a valid **CC** *Manipulate* operator constrains the initial state of the robot and object. Typically the contact must be a grasp that satisfies form closure. These criteria indicate that a desired motion of the robot will not cause it to release the object [16]. Such grasps may either be specified by the user or computed automatically[27]. It is also possible to constrain grasps with regard to robot force capabilities.

The advantages of **CC** are in interpretability and invariance to modeling error. The

disadvantage is in generality. We easily predict the possible displacements for an object by looking at robot and object geometry. Furthermore, an accurate geometric model is typically the only requirement for ensuring reliable execution after grasping. However, some objects, such as large boxes, are difficult or impossible to grasp. Constrained environments may also restrict robot positions to make grasping impossible or require the end-effector to move with respect to the object.

Pushing (Constrained Motion): Manipulation that does not require a grasp with closure is called non-prehensile [16]. *Constrained Motion* **CM** is a subclass of non-prehensile *Manipulate* operators. Given any contact between the robot and object **CM** restricts the path that the manipulator can follow in order to maintain a fixed transform between the end-effector and the object.

The most studied form of **CM** manipulation relies on static friction during pushing [15][28]. At sufficiently low velocities static friction prevents the object from slipping with respect to the contact surface. Given the friction coefficient and the friction center of the object we can restrict robot paths to those that apply a force inside the friction cone for the object.[15]

Constrained motion is more general than **CC** manipulation, however it relies on more detailed modeling. In addition to geometry, this method requires knowledge of friction properties. An incorrect assessment would lead to slip causing unplanned online behavior. **CM** is also less interpretable than **CC** since it is difficult to visualize the space of curves generated by local differential constraints.

Manipulation Primitives (MP): The *Manipulate* operator can be unconstrained in both grasp and motion. Morris and Morrow give taxonomies for possible interactions between a robot end-effector and an object [29, 30]. These include grasping/pushing and add other modes of interaction that allow translational or rotational slip.

Methods that do not guarantee a fixed transform between the robot and the object

rely on forward simulation of the object dynamics. Interaction with the robot is simulated to determine the displacement of the object. We experimented with **MP** in [31] by commanding translation and allowing slip in rotation. In some cases a parameterized **MP** yields solutions where grasping and pushing cannot.

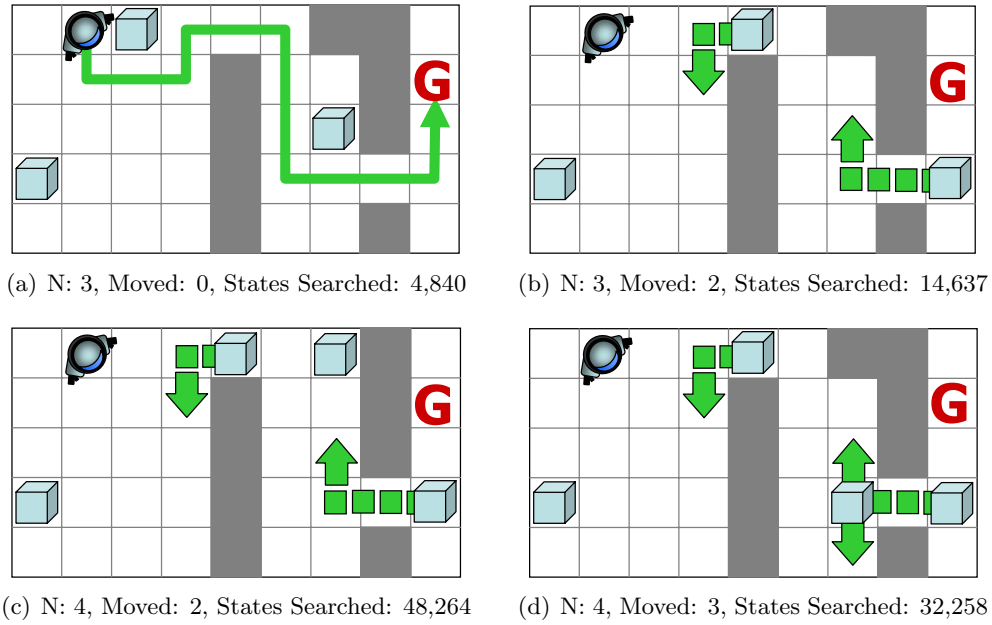
Further generality requires even more accurate dynamic modeling of the object for forward simulation. By allowing slip it is desirable that online sensing be used to close the loop and ensure that the object path is closely followed. The motions generated by constraining different degrees of freedom are not unique. Primitives can be restricted to a subset that spans a significant space of motions in a given environment.

All three classes of object motion interpret the robot trajectory as an object trajectory. We presented them in order of increasing generality and decreasing reliability. Intuitively, less constraint in contact and motion leads to greater demands on model accuracy. In theory one can combine these operators to achieve generality and stability whenever possible. Practically, *Manipulate* operators can be selected based on the geometry of the environment. A smaller set of permitted interactions restricts the possible displacements for the object and reduces the branching factor of search.

2.4 Complexity of Search

In contrast to a mobile robot, the branching factor of search for mobile manipulators relies on two modes of interaction. In addition to choosing paths for *Navigate*, the robot chooses contacts and paths for *Manipulate* operators. An even greater challenge to branching comes from the size of the state space. The robot must plan the state of the environment including all the positions of movable objects. Wilfong first showed that a simplified variant of this domain is NP-hard, making complete planning for the full domain computationally infeasible [32]. More recent work demonstrated NP-completeness results for trivial problems where square blocks can translated on a planar grid [33].

In order to better understand the source of complexity, consider a simplified grid



(a) N: 3, Moved: 0, States Searched: 4,840 (b) N: 3, Moved: 2, States Searched: 14,637

(c) N: 4, Moved: 2, States Searched: 48,264 (d) N: 4, Moved: 3, States Searched: 32,258

Figure 2.1: Simple NAMO problems on a planar grid.

world that contains a robot and m obstacles as shown in Figure 2.1. Suppose we attempt a complete search over the robot's action space. Let the configuration of the robot base be represented by $\mathcal{C}_R = (x, y)$, with resolution n in each direction. The generic size of the robot $\mathcal{C}$-space $|\mathcal{C}_R| = O(n^2)$. Analogously, for each object $|\mathcal{O}_i| = O(n^2)$. The full space of possible environment configurations is the product of these subspaces, $\mathcal{C}_R \times \mathcal{O}_1 \times \mathcal{O}_2 \times \ldots \times \mathcal{O}_N$, and therefore has $O(n^{2(N+1)})$ world states. The size of the search space is exponential in the number of objects that occupy the robot's environment.

The size of the configuration space relates directly to the complexity of a complete search over the robot's actions. In Figure 2.1, the robot can translate to an adjacent square, grasp an adjacent movable block and move while holding the block. The total number of possible actions at any time is 9: 4 directions of motion, 4 directions of grasping and one release. For simplicity, we assign equal cost to all actions and apply breadth-first search (BFS) to find a sequence of actions that gives the robot access to the goal.

In Figure 2.1(a), although no obstacles are moved, the search examines 4,840 distinct configurations before reaching the goal. Note that our search remembers and does

not revisit previously explored world states. While there are only $9 \times 5 = 45$ possible placements of the robot, every obstacle displacement generates a new world state where every robot position must be reconsidered. A change in the world configuration may open formerly unreachable directions of motion for the robot or for the objects.

To a human, Figure 2.1(c) may look more like Figure 2.1(b) than Figure 2.1(d). A previously static obstacle is replaced by a movable one. The additional movable obstacle is not in the way of reaching the goal. However, breadth-first search takes three times longer to find the solution. These results generalize from simple search to the most recent domain independent action planners. Junghanns [34] applied the *Blackbox*[35] planner to Sokoban problems where a robot pushes blocks on a grid. The AIPS planning competition winner showed a similar rise in planning time when adding only a single block.

The most successful planners for puzzle problems such as the 15-puzzle, Towers of Hanoi and Sokoban benefit significantly from identifying and pre-computing solutions to patterns of states. [36, 37, 34] In highly structured problems, these patterns can be recognized and used as heuristics or macros for search. Such methods are likely to also succeed in our grid world NAMO. However, our interest is in NAMO as a practical robot domain. Arbitrary object geometry and higher granularity of actions make it infeasible to pre-compute solutions to a significant portion of subproblems.

In the context of geometric planning, sampling based methods such as Probabilistic Roadmaps and Rapidly-Exploring Random Trees have been applied to problems with exponentially large search spaces. [38, 39] These methods are most effective in *expansive* spaces where it is easy to sample points that significantly expand the search tree. [40] NAMO problems typically have numerous narrow passages in the state space. In Figure 2.1, among thousands of explored actions only one or two object grasps and translations make progress to the goal. Section 3 will discuss the narrow passages that exist in NAMO planning and use them to guide search. Section 7 will apply sampling-based planning in NAMO domains with high dimensional action spaces such as those of articulated robots.

3

NAMO Planning

3.1 Overview

In Chapter 2 we defined the NAMO domain and noted the complexity of motion planning with movable obstacles. While complete planning for NAMO may be infeasible, this section gives a resolution complete algorithm for an intuitive subclass of problems. To arrive at this algorithm we first give a *configuration space* interpretation of our domain. We observe the inherent structure of environments that consist of disjoint components of free space. This observation leads to the definition of a linear class of problems and an abstract graph algorithm for solving them. Finally we give a practical variant of this algorithm, prove its validity and give experimental results in domains with nearly 100 movable objects.

3.2 Configuration Space

To understand the structure of NAMO and related spatial reasoning tasks, let us first interpret this problem in terms of the configuration space. [10] Let $\mathcal{C}_W$ be the space of all possible W^t. During a *Navigate* operation, the robot can only change its own configuration. Hence we denote a subspace or *slice* of $\mathcal{C}_W$:

$$\mathcal{C}_R(W^t) = (\{r\}, q_1^t, q_2^t, \ldots, q_m^t)$$

While $\mathcal{C}_R$ includes all possible configurations for the robot, some of them collide with static or movable obstacles. The free space of valid robot configurations is parameterized by the locations of movable obstacles. To make this relationship explicit let

$$A(q) = \{x \in \mathbb{R}^k | x \text{ is a point of object } A \text{ in configuration } q\}. \tag{3.2.1}$$

For any set of obstacle points S in $\mathbb{R}^k$, a configuration space obstacle in $\mathcal{C}_A$ is the set: $\mathcal{X}_A(S) = \{q \in \mathcal{C}_A | A(q) \cap S \neq \emptyset\}$. Let q be a configuration of A and p be a configuration of object B. Since two objects cannot occupy the same space in $\mathbb{R}^n$, $\mathcal{X}$ is symmetric:

$$p \in \mathcal{X}_B(A(q)) \Rightarrow B(p) \cap A(q) \neq \emptyset \Rightarrow q \in \mathcal{X}_A(B(p)) \tag{3.2.2}$$

To simplify notation we define the following: $\mathcal{X}_R^{O_i}(W^t) = \mathcal{X}_R(O_i(q_i^t))$ and $\mathcal{X}_{O_j}^{O_i}(W^t) = \mathcal{X}_{O_j}(O_i(q_i^t))$ represent obstacles due to O_i in $\mathcal{C}_R$ and $\mathcal{C}_{O_j}$ respectively. For any set of obstacle points S in $\mathbb{R}^k$, a configuration space obstacle in $\mathcal{C}_R$ is the set: $\mathcal{X}_R(S) = \{r \in \mathcal{C}_R | R(r) \cap S \neq \emptyset\}$. Lastly, $\overline{\mathcal{X}_A(B)}$ is the complement of $\mathcal{X}_A(B)$ in $\mathcal{C}_A$.

The free space of a robot, $\mathcal{C}_A^{free}(W^t)$, is the set of configurations where the object is not in collision with fixed or movable obstacles. Eq. 3.2.3 defines $\mathcal{C}_R^{free}(W^t)$ and $\mathcal{C}_{O_i}^{free}(W^t)$ as sets of collision-free configuration for the robot and movable objects. Figure 3.1(a) shows the free configuration space $\mathcal{C}_R^{free}(W^t)$ for a circular robot.

$$\mathcal{C}_R^{free}(W^t) = \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_i \overline{\mathcal{X}_R^{O_i}(W^t)} \qquad \mathcal{C}_{O_i}^{free}(W^t) = \bigcap_k \overline{\mathcal{X}_{O_i}(F_k)} \bigcap_{O_j \neq O_i} \overline{\mathcal{X}_{O_i}^{O_j}(W^t)} \tag{3.2.3}$$

We can use the $\mathcal{C}$-space representation to identify valid *Manipulate* and *Navigate* operators. *Navigate* is valid if and only if its path is collision free:

$$\tau(s) \in \mathcal{C}_R^{free}(W^t) \qquad \forall s \in [0,1] \tag{3.2.4}$$

Eq. 3.2.5 - 3.2.9 validate a *Manipulate* operator. In addition to satisfying collision-free motion manipulation must end with the object in a statically stable *placement*

$\mathcal{C}_{O_i}^{place}(W^t) \subseteq \mathcal{C}_{O_i}^{free}(W^t)$ (3.2.8). Eq. 3.2.9 ensures that the robot does not collide with obstacle O_i. In our two dimensional examples, we assume gravity is orthogonal to the object plane and hence $\mathcal{C}_{O_i}^{place}(W^t) = \mathcal{C}_{O_i}^{free}(W^t)$.

$$\tau(s) \in \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_{j \neq i} \overline{\mathcal{X}_R^{O_j}(W^t)} \qquad \forall s \in [0,1] \tag{3.2.5}$$

$$\tau_{O_i}(s) \in \mathcal{C}_{O_i}^{free}(W^t) \qquad \forall s \in [0,1] \tag{3.2.6}$$

$$\tau_{O_i}(0) = q_i^t \tag{3.2.7}$$

$$\tau_{O_i}(1) \in \mathcal{C}_{O_i}^{place}(W^t) \tag{3.2.8}$$

$$R(\tau(s)) \cap O_i(\tau_{O_i}(s)) = \emptyset \qquad \forall s \in [0,1] \tag{3.2.9}$$

3.3 Goals for Navigation

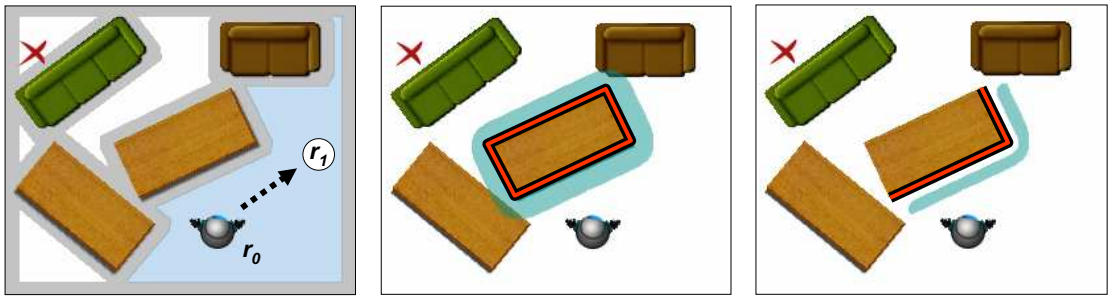
Having defined our domain in terms of configuration space we can begin to make useful observations about the planning problem. So far, we have looked at *Manipulate* and *Navigate* operators independently. However, the most interesting aspect of NAMO is the interdependence of actions. For instance, in order for the robot to manipulate an object it must be within reach of the object. It is not always possible to make contact with an object without previously moving another one. In this section we define *non-trivial* goals for navigation and use them to constrain subsequent manipulation.

First, consider the two robot configurations depicted in Figure 3.3(a). There exists a collision free path $\tau(r_0, r_1)$ that validates $Navigate(W^0, \tau(r_0, r_1))$. Finding this path is a typical problem for existing motion planners. Eq. 3.3.1 generalizes this relationship to all robot configurations that are accessible from r^t in one step of *Navigate*.

$$\mathcal{C}_R^{acc}(W^t) = \{r_j \in \mathcal{C}_R^{free}(W^t) | \text{ exists } \tau(r^t, r_j) \text{ s.t. } \forall s(\tau[s] \in \mathcal{C}_R^{free}(W^t)\} \tag{3.3.1}$$

Furthermore, the configurations in $\mathcal{C}_R^{acc}(W^t)$ can all be reached from one another.

$$\forall r_i, r_j \in \mathcal{C}_R^{acc}(W^t) \text{ exists } \tau(r_i, r_j) \text{ s.t. } \forall s(\tau[s] \in \mathcal{C}_R^{acc}(W^t)) \tag{3.3.2}$$



(a) $A, B \in \mathcal{C}_R^{acc}(W^0)$ (b) $\mathbf{G}(O_i) \rightarrow \mathcal{C}_R^{cont}(O_i, W^t)$ (c) $\mathcal{C}_R^{acc}(W^t) \cap \mathcal{C}_R^{cont}(O_i, W^t)$

Figure 3.1: Simulated 2D NAMO $\mathcal{C}_R$-space for a circular robot.

The space of accessible configurations is lightly shaded in Figure 3.1(a). We can compute configurations that belong to this set using grid-based wavefront propagation.[41]

Accessible configurations are feasible goals for navigation. Likewise, contact configurations are feasible starting states for manipulation. Any class of *Manipulate* operators in Section 2.3 restricts the initial robot configuration such that robot motion will result in the displacement of the object. For instance, form closure requires the end-effector to surround part of the object and pushing demands surface contact.

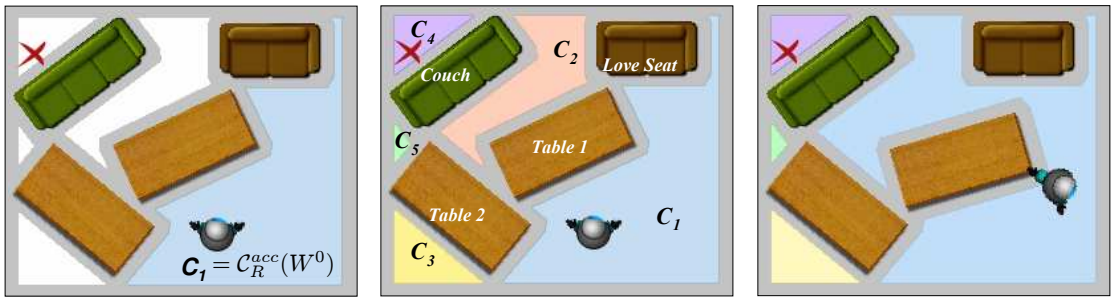
We defined valid contacts $\mathbf{G}(O_i)$ as a set of end-effector positions relative to O_i. Given the object configuration, q_i^t, this set maps to absolute positions for the end-effector. Absolute positions map to robot configurations via inverse kinematics (IK).

$$\mathcal{C}_R^{cont}(O_i, W^t) = IK(\mathbf{G}(O_i), W^t) \tag{3.3.3}$$

In Figure 3.1(b) the table can be grasped at any point on the perimeter. The shaded area represents $\mathcal{C}_R^{cont}(Table_1, W^0)$ as a set of feasible positions for the robot base that make it possible to grasp the object. Figure 3.1(c) illustrates the intersection of $\mathcal{C}_R^{acc}$ and $\mathcal{C}_R^{cont}$. These configurations are object contacts that are accessible to the robot in W^t.

$$\mathcal{C}_R^{AC}(O_i, W^t) = \mathcal{C}_R^{acc}(W^t) \bigcap \mathcal{C}_R^{cont}(O_i, W^t) \tag{3.3.4}$$

The set $\mathcal{C}_R^{AC}(O_i, W^t)$ represents useful goals for the *Navigate* operator in W^t. There are



(a) $C_1 = \mathcal{C}_R^{acc}(W^0)$ (b) $\mathcal{C}_R^{free}$ Components (c) Keyhole Solution

Figure 3.2: NAMO $\mathcal{C}_R$-space partitioned into components.

two cases. When $\mathcal{C}_R^{acc}(W^t)$ contains the goal state, NAMO reduces to a path plan to the goal. Otherwise, at least one object must be manipulated prior to reaching the goal. Since *Navigate* only displaces the robot, $\mathcal{C}_R^{acc}(W^t)$ and $\mathcal{C}_R^{cont}(O_i, W^t)$ do not change after the operation for any O_i. By definition $\mathcal{C}_R^{AC}(O_i, W^t)$ is not affected. $Navigate(W^t, \tau(r^t, r^{t+1}))$ must satisfy $r^{t+1} \in \mathcal{C}_R^{AC}(O_i, W^t)$ for some O_i in order to make progress.

3.4 Goals for Manipulation

In Section 3.3 we saw that the contact states for manipulation constrain useful goals for *Navigate*. We also know that the subsequent *Manipulate* operation can only be applied from one of these states to one of the accessible objects. In this section we look at how the navigation space can be used to select goals for manipulation. These concepts form the basis for our first planner in the domain of Navigation Among Movable Obstacles.

In general, there is no method for identifying non-trivial manipulation. Unlike *Navigate*, any valid *Manipulate* operator changes the state of the world and the set of accessible configurations. Even if the change seems to decrease immediate accessibility it is possible that manipulation opens space for a future displacement of another object. Some manipulation actions, however, are more clearly useful than others. To identify them, let us return to the sets of configurations defined in Section 3.3.

We already observed that $\mathcal{C}_R^{acc} \in \mathcal{C}_R^{free}$ is a subspace of configurations that can be reached from one another by a single *Navigate* action. Suppose that the robot was in a

free configuration outside of $\mathcal{C}_R^{acc}$. There would also be a set of configurations accessible to the robot. In fact, as shown in Figure 3.2(b), we can partition the robot free space, $\mathcal{C}_R^{free}$, into disjoint sets of robot configurations that are closed under *Navigate* operators: $\{C_1, C_2, \ldots, C_d\}$. The goal configuration lies in one of these subsets, C_G.

Partitioning the free space results in an abstraction for identifying useful manipulation subgoals. Consider the effect of *Manipulate* in Figure 3.2(c). After the action, configurations in C_2 become valid goals for *Navigate*. The illustration shows part of the solution to a *keyhole* in this NAMO problem.

Definition 3.4.1. A *keyhole*, $K(W^1, C_i)$, is a subgoal problem in NAMO that specifies a start state, W^1, and a component of free space, C_i. The goal is to find a sequence of operators that results in W^2 s.t. every free configuration in C_i is accessible to the robot:

$$C_i \cap \mathcal{C}_R^{free}(W^2) \subset \mathcal{C}_R^{acc}(W^2) \qquad \text{and} \qquad C_i \cap \mathcal{C}_R^{free}(W^2) \neq \emptyset \tag{3.4.1}$$

To restrict the number of obstacles moved in solving a keyhole we define a *keyhole solution* according to the maximum number of permitted *Manipulate* operators.

Definition 3.4.2. A *k-solution* to $K(W^1, C_i)$ is a sequence of valid actions including at most k *Manipulate* operators that results in W^2 satisfying Eq. 3.4.1.

Manipulating obstacles to solve keyholes is useful because it creates passages in the robot's navigation space which open to entire sets of valid navigation goals. Rather than considering individual actions, we can simplify a NAMO task as follows: *The robot must resolve a sequence of keyholes until* $r^T \in C_G$ *at some future time* T.

3.5 Planning as Graph Search

With the introduction of useful subgoals for *Navigate* and *Manipulate* operators we now describe a conceptual planner that applies these goals to solving NAMO. First, we present an intuitive class of *linear* problems for which the planner is resolution complete. Next we describe the local search method used by our planner and give the planning algorithm.

3.5.1 Linear Problems

NAMO problems have numerous movable obstacles and exponentially as many world states. However, typically the number of free space components is significantly smaller. In this section we identify a class of problems that reduces the complexity of NAMO to choosing and solving a sequence of keyholes.

Notice that the solution to one keyhole may interfere with solving another by occupying or blocking parts of $\mathcal{C}_{free}$. Taking into account the history of how a robot entered a given free space component returns planning complexity to a search over all obstacle displacements. We therefore introduce a class of problems where keyholes can be solved independently.

Definition 3.5.1. A NAMO problem, (W_0, r^f), is *linear of degree k* or L_k if and only if:

1. There exists an ordered set of $n \geq 1$ distinct configuration space components, $\{C_1, \ldots, C_n\}$ such that $C_n = C_G$ and $C_1 = \mathcal{C}_R^{acc}(W_0)$.
2. For any $i (0 < i < n)$, any sequence of *k-solutions* to $\{K(W_{j-1}, C_j) | j < i\}$ results in W_{i-1} such that there exists a *k-solution* to $K(W_{i-1}, C_i)$.
3. If $n > 1$, any sequence of *k-solutions* to $\{K(W_{j-1}, C_j) | j < n\}$ results in W_{n-1} s.t. there exists a *k-solution* to $K(W_{n-1}, C_n)$ that results in W_n where $r^f \in \mathcal{C}_R^{free}(W_n)$.

For at least one sequence of free space components this inductive definition ensures that once the robot has reached a configuration in C_i it can always access C_{i+1}. Condition (3) guarantees the existence of a final keyhole solution such that the goal is in $C_G \cap \mathcal{C}_R^{free}(W^n)$. Although this definition allows for arbitrary k, this chapter we will focus on problems that are *linear of degree 1* or L_1.

Even when only one *Manipulation* operator is permitted, it is often possible to find a keyhole solution that blocks a subsequent keyhole. For instance, in Figure 3.2(c), consider turning the table to block a future displacement of the couch. We propose to broaden the scope of problems that can be represented as L_1 by constraining the action space of the robot. We have considered two restrictions on the placement of objects:

Occupancy Bounds: Any displacement of an object must not occupy additional free space that is not accessible to the robot: $\mathcal{X}_R^{O_i}(W^{t+1}) \subset (\mathcal{X}_R^{O_i}(W^t) \cup \mathcal{C}_R^{acc}(W^t))$. This implies that any solution to (W^{t-1}, C_t) results in W^t s.t. $C_t \cap \mathcal{C}_R^{free}(W^t) = C_t$.

Minimal Intrusion: Paths used for *Manipulate* operators are restricted to being minimal with respect to a chosen criterion. The criteria could include the dimension of inaccessible $\mathcal{C}_R$ occupied by a displaced object, the distance of the displacement or a simulation of expected work to be used in manipulation.

Occupancy bounds create an intuitive classification for problems. If there is a sequence of free space components where each C_i can be accessed using only the space of C_{i-1} then the problem is *linear*. Minimal intrusion is less intuitive since it requires some notion of the extent to which a previous component could be altered and still yield sufficient space for a keyhole solution. However, this method is more powerful since it allows the robot to take advantage of space in both C_i in addition to C_{i-1} for placing objects.

Regardless of restrictions, L_1 includes many real world problems. Generally, $\mathcal{C}_R^{free}$ is connected and allows navigation. L_1 problems arise when the state is perturbed due to the motion of some object. An L_1 planner should detect this object and restore connectivity.

3.5.2 Local Manipulation Search

The solution to a linear NAMO problem requires us to determine a sequence of keyholes that connect free space components. First, we introduce a simple search routine for accessing $C_2 \subset \mathcal{C}_R^{free}$ from $r^t \in C_R^{acc}(W^t)$ by moving a single object O_i. The routine returns W^{t+2}, a robot path, τ_M, and the total cost c or NIL when no path is found.

MANIP-SEARCH(W^t, O_i, C_2): We apply Section 3.3 to find a discrete or sampled set of contact states for *Manipulate*: $\mathcal{C}_R^{AC}(W^t)$. Starting from all distinct configurations $r^{t+1} = G_i$ we perform a uniform cost (breadth first) search over paths τ_M that validate: $Manipulate(W^t, O_i, G_i, \tau_M(r^{t+1}, r^{t+2}))$. The object path τ_o is determined by the specific action space. The search terminates when W^{t+2} satisfies Eq. 3.4.1 or every reachable state has been examined. If $r^f \in C_2$ the search must also ensure that $r^f \in C_R^{free}(W^{t+2})$.

Local uniform cost search allows the specification of any cost function and ensures that the returned path will minimize this function over the discrete action set. When restricting the action space to minimize intrusion (3.5.1) the cost should reflect the intrusion criteria such as occupied inaccessible space. We have optimized for path length in quasi-static planning and work or energy usage for dynamic manipulation.

Although MANIP-SEARCH is a simple method, efficient recognition of goal satisfaction is non-trivial. We provide one possible solution in Appendix A.

3.5.3 Connecting Free Space

Given a routine for accessing a component of free space we turn to the global task of solving linear problems. In this section we introduce the global CONNECTFS algorithm.

First, define a set **FC** of disjoint free space components $C_i \subset \mathcal{C}_R^{free}$. C_S and C_G refer to the components containing the robot and the goal respectively. Our algorithm keeps a search tree, **FCT**, and a queue of unexpanded nodes **Q**. Each node is a pair (C_i, W^t) and each edge is an obstacle O_l. **FCT** and **Q** are initialized with the node (C_S, W^0). At each step we remove a node from **Q**, $N = (C_i, W^t)$ and expand the tree as follows:

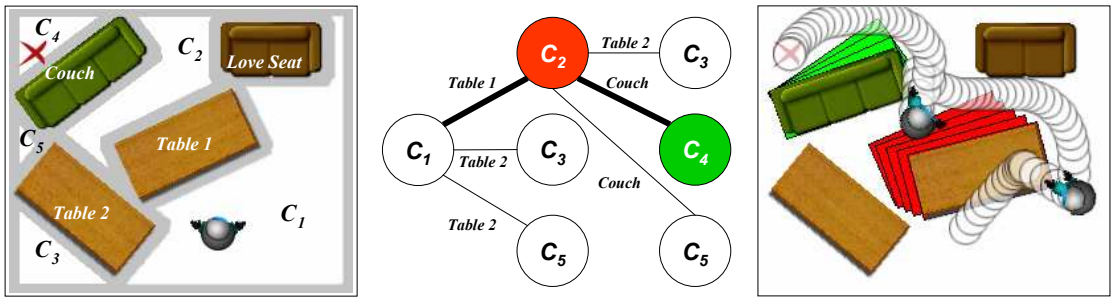
1. Construct the set $\mathbf{C}_N \subset \mathbf{FC}$ of all free space components, C_j, such that (C_j, W') is not an ancestor of N in **FCT** for any W'.

2. For each $C_j \in \mathbf{C}_N$ find the set of *neighboring* obstacles $\mathbf{O}_N(C_j)$ such that there exists a path from r^t to some r_j in C_j that collides only with the obstacle:

$$\mathbf{O}_N(C_j) = \{O_l | \exists r_j \in C_j, \tau(r^t, r_j) : \forall s(\tau[s] \in \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathcal{X}_R^{O_j}(W^t)})\} \quad (3.5.1)$$

3. Remove $C_j \in \mathbf{C}_N$. For each $O_l \in \mathbf{O}_N(C_j)$ let $S(O_l) = (W^{t+2}, \tau, c)$ be the result of calling MANIP-SEARCH(W^t, O_l, C_j). If at least one call succeeds choose $S(O_l)$ with least cost and add the node $N' = (C_j, W^{t+2})$, edge $e(N, N') = O_l$ to **FCT**.

4. Repeat step 3 until $\mathbf{C}_N = \emptyset$.



(a) L_1 NAMO Problem (b) **FCT** with solution path (c) Solution

Figure 3.3: Construction of FCT structures the solution for L_1 problems.

We repeat the expansion of nodes until $N^G = (C_G, W^T)$ is added to **FCT** for some W^T or every node has been removed from **Q**.

3.5.4 Analysis

Our description of ConnectFS illustrates the potential for using a configuration space decomposition of NAMO problems to select subproblems that can easily be solved by a motion planner. Furthermore, the construction of ConnectFS allows for a simple proof of its relationship to the L_1 problem class. First, we must show that any *1-solution* to a keyhole must move a *neighboring* object as defined in step 2.

Lemma 3.5.2. *If there exists a* 1-solution *to* $K(W^T, C_j)$ *which displaces* O_l *then there exist* $r_j \in C_j$ *and* $\tau(r^T, r_j)$ *such that for all* s, $\tau[s] \in \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathcal{X}_R^{O_j}(W^T)}$.

Proof. To simplify notation let $\mathbf{A}^t = \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathcal{X}_R^{O_j}(W^t)}$. Assume there exists a *1-solution* consisting of a sequence of n operators, each parameterized by a robot path $\tau(r^t, r^{t+1})$ where t refers to any time step such that $(T \leq t \leq T+n)$.

Since only O_l is displaced between T and $T+n$, all other obstacle configurations are unchanged: $q_j^t = q_j^T$ for all $j \neq l$ and t. Hence, $\mathbf{A}^t = \mathbf{A}^T$ for all t. We now show that every operator path in the *1-solution* satisfies $\tau[s] \in \mathbf{A}^T$ for all s.

Manipulate$(W^t, O_l, G_i, \tau_M(r^t, r^{t+1}))$: The operator is valid only if the path satisfies $\tau_M[s] \in \mathbf{A}^t$ for all s (3.2.5). Hence, the path satisfies $\tau_M[s] \in \mathbf{A}^T$ for all s.

Navigate$(W^t, \tau_N(r^t, r^{t+1}))$: The operator is valid only if the path satisfies $\tau_N[s] \in \mathcal{C}_R^{free}(W^t)$ for all s (3.2.4). Since $\mathcal{C}_R^{free}(W^t) = \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_j \overline{\mathcal{X}_R^{O_j}(W^t)}$, we have $\mathcal{C}_R^{free}(W^t) \subset \mathbf{A}^t$. Therefore, $\tau_N[s] \in \mathbf{A}^t$ and consequently $\tau_N[s] \in \mathbf{A}^T$ for all s.

Furthermore, the definition of a *1-solution*, Eq. 3.4.1, guarantees the existence of $r_j \in C_j \cap \mathcal{C}_R^{free}(W^{T+n})$. Since $C_j \cap \mathcal{C}_R^{free}(W^{T+n}) \subset \mathcal{C}_R^{acc}(W^{T+n})$, there exists $\tau_j(r^{T+n}, r_j)$ such that $\tau_j[s] \in \mathcal{C}_R^{free}(W^{T+n})$ for all s(Eq. 3.3.1). Therefore $\tau_j[s] \in \mathbf{A}^{T+n} = \mathbf{A}^T$ for all s.

Finally, we construct a path $\tau_D(r^T, \ldots, r^{T+n}, r_j)$ by composing the operator paths from the *1-solution* with τ_j. Since $\tau[s] \in \mathbf{A}^T$ for all s in all subpaths of τ_D, we also have $\tau_D[s] \in \mathbf{A}^T$ for all s. □

Lemma 3.5.3. ConnectFS *is resolution complete for problems in* L_1.

Proof. Let Π be a solvable problem in L_1. We show that ConnectFS will find a solution. By definition of L_1, there exists an ordered set Ω of n disjoint $\mathcal{C}_R^{free}$ components $\{C_S, \ldots, C_G\}$. We show that ConnectFS will add (C_G, W^T) to **FCT** by induction. In the base case, $(C_S, W^0) \in$ **FCT**.

Assume (C_{i-1}, W^t) has been added to **FCT** such that $C_{i-1} \in \Omega, i < n$ and W^t was produced by a sequence of *1-solutions* to $\{K(W_{j-1}, C_j) | 0 < j < i\}$. By the definition of L_1 there exists a *1-solution* with n operators to $K(W^t, C_i)$ (Eq. 3.4.1). Hence, starting in W^t, there is an operator sequence with one *Manipulate* that displaces some obstacle O_l and results in W_i such that $C_i \cap \mathcal{C}_R^{free}(W_i) \subset \mathcal{C}_R^{acc}(W_i)$.

When (C_{i-1}, W^t) is expanded, Lemma 3.5.2 guarantees that there exist $r_i \in C_i$ and a path $\tau_D(r^t, r_i)$ in W^t that only passes through O_l. Consequently, in step 2 of ConnectFS, O_l will be added to $\mathbf{O}_N(C_i)$. Step 3 will call $W^{t+2} =$ Manip-Search(W^t, O_l, C_i). Since Manip-Search is resolution complete over the action space, it will find a *Manipulate* path that satisfies Eq. 3.4.1. Therefore, (C_i, W^{t+2}) will be added to **FCT**.

By induction, (C_G, W^T) will be added to **FCT**. Regarding termination, let d be the number of free space components in **FC**. Since all nodes added to the tree must contain distinct free space components from their ancestors the tree has a maximum depth of d

(step 1). Furthermore, each node at depth $i (1 \leq i \leq d)$ has at most $d - i$ children. Either (C_G, W^T) is added or all nodes are expanded to depth d and no further nodes can be added to $\mathbf{Q}$. Hence ConnectFS terminates in finite time and is resolution complete for problems in L_1. □

The construction of **FCT** in ConnectFS also allows us to optimize for various criteria. First of all, we can find solutions that minimize the number of objects moved simply by choosing nodes from $\mathbf{Q}$ according to their depth in the tree. Nodes of lesser depth in the tree correspond to less displaced objects. We can also associate a cost with each node by summing the cost of the parent node and the cost returned by Manip-Search. When a solution of cost c is found, we would continue the search until all nodes in $\mathbf{Q}$ have cost $c' > c$. The lowest cost solution would be minimal with respect to Manip-Search costs.

3.5.5 Challenges of ConnectFS

Although ConnectFS reduces the search space from brute force action-space search, its primary purpose is to convey the utility of the reduced dimensional configuration space structure. Practically, a NAMO domain could have large numbers objects and free-space components. Constructing the tree would require an algorithm to determine all *neighboring* objects for all components of free-space. Furthermore, we would need to call Manip-Search to verify every potential access. This seems unreasonable for a navigation planner since we may only need to move one or two objects to reach the goal.

We believe that this observation is critical for developing a real-time system. In particular, the complexity of the problem should depend on the complexity of resolving *keyholes* that lie on a reasonable navigation path, not on the complexity of unrelated components of the world. In other words, *since the purpose of NAMO is navigation, a NAMO problem should only be difficult when navigation is difficult.*

Section 3.6 gives one answer to this challenge. We introduce a heuristic algorithm that implicitly follows the structure of ConnectFS without graph construction. To do so, we again turn to the navigational substructure of the problem. Previously, we observed that

every L_1 plan is a sequence of actions that access entire components of $\mathcal{C}_R^{free}$. Lemma 3.5.2 also showed that obstacles that lie in the path of *Navigate* should be considered for motion. We will now consider extending such paths through the rest of $\mathcal{C}_R$ until they reach the goal and using them to guide the planner in subgoal selection.

3.6 Planner Prototype

With the results from the previous sections, we now formulate a simple and effective planner for the NAMO domain: SelectConnect. The planner is a best-first search that generates fast plans in L_1 environments. Its heuristic, RCH, is a navigation planner with relaxed constraints. RCH selects obstacles to consider for motion. Following the algorithm description, we show that its search space is equivalent to that of ConnectFS (Section 3.5.3). With best-first search, optimality is no longer guaranteed. However, the planner is much more efficient and resolution complete for problems in L_1. If memory and efficiency are less of a concern, optimality can be regained with uniform-cost search.

3.6.1 Relaxed Constraint Heuristic

The Relaxed Constraint Heuristic (RCH) is a navigation planner that allows collisions with movable obstacles. It selects obstacles for SC to displace. RCH is parameterized by W^t, *AvoidList* of (O_i, C_i) pairs to avoid, and *PrevList* of visited C_j. After generating a path estimate to the goal, it returns the pair (O_1, C_1) of the first obstacle in the path, and the first component of free space. If no valid path exists, RCH returns NIL. Since RCH does not move obstacles, all obstacle positions are given in terms of W^t.

RCH is a modified A^* search from r^t to r^f on a dense regular grid of $\mathcal{C}_R(W^t)$ which plans *through* movable obstacles. RCH keeps a priority queue of grid cells $\mathbf{x}_i$. Each cell is associated with a robot configuration, r_i, the cost of reaching it, $g(\mathbf{x}_i)$, and the estimated cost for a solution path that passes through r_i, $f(\mathbf{x}_i)$. On each iteration, RCH removes $\mathbf{x}_i$ with least $f(\mathbf{x}_i)$ from the queue and *expands* it by adding valid adjacent cells $\mathbf{x}_j$ with the costs in Eq. 3.6.1. Let $e(r_i, r_j)$ be the transition cost of entering a cell $r_j \in \mathcal{X}_R^{O_j}$ from

RCH($W^t, AvoidList, PrevList, r^f$)
1 $Closed \leftarrow \emptyset$
2 $Q \leftarrow$ MAKE-PRIORITY-QUEUE($r^t, 0, 0$)
3 **while** $Q \neq$ **empty**
4 **do** $\mathbf{x}_1 = (r_1, O_F, C_F) =$ REMOVE-FIRST(Q)
5 **if** ($\mathbf{x}_1 \in Closed$) **continue**
6 **if** ($r_1 = r^f$ **and** $O_F \neq 0$ **and** $C_F \neq 0$) **return** (O_F, C_F))
7 $Closed$ **append** $(\mathbf{x}_1)$
8 **for each** $r_2 \in$ ADJACENT(r_1)
9 **do if** ($C_F \neq 0$) ENQUEUE($Q, (r_2, O_F, C_F)$); **continue**
10 **if** ($O_F \neq 0$ **and** $r_2 \in \mathcal{C}_R^{free}$) ENQUEUE($Q, (r_2, O_F, 0)$)
11 **if** ($O_F \neq 0$ **and** $r_2 \in$ **exc** $\mathcal{X}_R^{O_F}$)ENQUEUE($Q, (r_2, O_F, 0)$)
12 **if** ($O_F \neq 0$ **and** $r_2 \in C_i$ **s.t.** $C_i \notin PrevList$ **and** $(O_F, C_i) \notin AvoidList$)
13 ENQUEUE($Q, (r_2, O_F, C_i)$)
14 **if** ($O_F = 0$ **and** $r_2 \in \mathcal{C}_R^{free}$) ENQUEUE($Q, (r_2, 0, 0)$)
15 **if** ($O_F = 0$ **and** $r_2 \in$ **exc** $\mathcal{X}_R^{O_i}$) ENQUEUE($Q, (r_2, O_i, 0)$)
16 **return** NIL

Figure 3.4: Pseudo-code for RCH.

$r_i \notin \mathcal{X}_R^{O_j}$, where e is estimated from the size or mass of O_j. $h(x_j, x^f)$ estimates goal distance and α relates the estimated cost of moving an object to the cost of navigation.

$$g(\mathbf{x}_j) = g(\mathbf{x}_i) + (1 - \alpha) + \alpha e(r_i, r_j)$$
$$f(\mathbf{x}_j) = g(\mathbf{x}_j) + h(x_j, x^f) \tag{3.6.1}$$

As shown in Figure 3.4, RCH restricts valid transitions. We use *exclusively contained* to refer to r_i contained in only one obstacle: $r_i \in$ **exc** $\mathcal{X}_R^{O_i} \Rightarrow (O_j \neq O_i$ then $r_i \notin \mathcal{X}_R^{O_j})$.

Definition 3.6.1. A path $P = \{r_1, \ldots, r_n = r^f\}$ is $Valid(O_i, C_j)$ if and only if $(O_i, C_j) \notin AvoidList$ and $O_j \notin PrevList$ and the path collides with exactly one obstacle prior to entering C_j. Alternatively, there exists $m < n$ such that: $r_m \in C_j$ and for all r_i where $i < m$ either $r_i \in \mathcal{X}_R^{O_i}$ or $r_i \in \mathcal{C}_R^{free}$.

Since the validity of a transition depends on the history of objects/free-space encountered along a path, we expand the state with a dimension for each obstacle and $\mathcal{C}_R^{free}$ component. The states searched are triples $\mathbf{x}_i = (r_i, O_i, C_j)$. O_i is the first obstacle encountered or 0 if one has not been encountered. C_j is likewise 0 or the first free space component.

Lemma 3.6.2. *If there exists a $Valid(O_F, C_F)$ path then RCH will find a solution.*

Proof. Assume there exists a $Valid(O_F, C_F)$ path P for some O_F, C_F then the RCH will find a path. After adding $(r^t, 0, 0)$ to Q, line 14 expands the children of this state adding all $r_i \in \mathcal{C}_R^{acc}$ to Q as $(r_i, 0, 0)$. Line 15 allows the transition from any $(r_i, 0, 0)$ to any adjacent $(r_j, O_A, 0)$ where $r_j \in \mathcal{X}_R^{O_A}$. Lines 10 and 11 expand any state $(r_i, O_A, 0)$ to $(r_j, O_A, 0)$ where $r_j \in \mathcal{C}_R^{free}$ or r_j is exclusively in O_A. Hence every state that can be reached after entering only one obstacle will be added to Q as $(r_i, O_A, 0)$.

From Def. 3.6.1 we know that there exists such a state $(r_{m-1}, O_F, 0)$ in P that can be reached after entering only one obstacle. Furthermore, when $(r_{m-1}, O_F, 0)$ is expanded to r_m the conditions on line 12 will be satisfied since $r_m \in C_F$ and $C_F \notin PrevList$ and $(O_F, C_F) \notin AvoidList$. Hence (r_m, O_F, C_F) will be added to Q on line 13. Finally, line 9 will continue to expand this state to all adjacent states as (r_i, O_F, C_F). Since path P exists, there exists some path from r_m to the goal and therefore (r^f, O_F, C_F) will be found. We know the process will terminate since states are not revisited by addition to *Closed* and the sizes of $\{r_i\}, \{O_i\}$ and $\{C_i\}$ are finite. □

Lemma 3.6.3. *If RCH finds a solution then there exists a $Valid(O_F, C_F)$ path.*

Proof. Suppose RCH has found a solution path P terminating in (r_j, O_F, C_F) on line 6. Lines 10-15 do not permit any transition from a state (r_j, O_F, C_F) to $(r_i, O_F, 0)$ or from $(r_j, O_F, 0)$ to $(r_i, 0, 0)$. Consequently we can separate P into three segments. P_1 consists of states $(r_i, 0, 0)$, P_2 contains $(r_i, O_F, 0)$ and P_3 contains (r_i, O_F, C_F). Any transition from the last state in P_2 to the first in P_3 must have satisfied the condition on line 12, hence $C_F \notin PrevList$ and $(O_F, C_F) \notin AvoidList$. Furthermore, every state in P_2 must be either in $\mathcal{C}_R^{free}$ or exclusively in $\mathcal{X}_R^{O_F}$ as added by lines 13,10 and 11. Finally, every state in P_1 must be in $\mathcal{C}_R^{free}$ since it was added by line 15. Hence the path P satisfies the second criterion of Def. 3.6.1. □

While this algorithm appears more complex than A^* the branching factor is unchanged. Furthermore, only objects that can be reached without colliding with other objects are

```
SELECTCONNECT(W^t, PrevList, r^f)
 1  AvoidList ← ∅
 2  if x^f ∈ C_R^acc(W^t)
 3     then return (FIND-PATH(W^t, x^f))
 4  while (O_1, C_1) ← RCH(W^t, AvoidList, PrevList, r^f) ≠ NIL
 5  do
 6     (W^{t+2}, τ_M, c) ← MANIP-SEARCH(W^t, O_1, C_1)
 7     if τ_M ≠ NIL
 8        then FuturePlan ← SELECTCONNECT(W^{t+2}, PrevList append C_1, r^f)
 9             if FuturePlan ≠ NIL
10                then τ_N ← FIND-PATH(W^t, τ_M[0])
11                     return ((τ_N, τ_M) append FuturePlan)
12     AvoidList append (O_1, C_1)
13  return NIL
```

Figure 3.5: Pseudo-code for SELECTCONNECT.

taken into account. To increase efficiency, membership in *Closed* on line 6 can be checked using (r_i, O_i) rather than the full state. Since there are no restrictions on transitions from any non-zero C_i path existence will not depend on its value.

3.6.2 High Level Planner

Figure 3.6.2 gives the pseudo-code for SELECTCONNECT (SC). The planner makes use of RCH and MANIP-SEARCH, as described in Section 3.5.2. It is a greedy heuristic search with backtracking. The planner backtracks locally when the object selected by RCH cannot be moved to merge the selected $C_i \subset \mathcal{C}_{free}$. It backtracks globally when all the paths identified by RCH from C_i are unsuccessful.

SC calls FIND-PATH to determine a *Navigate* path from r^t to a contact, r^{t+1}. The existence of $\tau_N(r^t, r^{t+1})$ is guaranteed by the choice of contacts in MANIP-SEARCH.

3.6.3 Examples and Experimental Results

We have implemented the proposed NAMO planner in a dynamic simulation environment. The intuitive nature of SELECTCONNECT is best illustrated by a sample problem solution generated by the planner. In Figure 3.6(a), we see that $\mathcal{C}_R^{free}$ is disjoint - making this a

(a) Problem (b) RCH (c) Keyhole Solution (d) Final Plan

Figure 3.6: Walk-through of an autonomous SELECTCONNECT.

NAMO problem. Line 4 of SC calls RCH, the heuristic sub-planner. RCH finds that the least cost $Valid(O_i, C_j)$ path to the goal lies through $O_i = Couch$. The path is shown in Figure 3.6(b). RCH also determines that the free-space component to be connected contains the goal. Line 6 calls MANIP-SEARCH to find a motion for the couch. Figure 3.6(c) shows the minimum cost manipulation path that opens the goal free-space. Finally, SELECTCONNECT is called recursively. Since r^f is accessible, line 3 finds a plan to the goal and completes the procedure (Figure 3.6(d)). The remainder of the pseudo-code iterates this process until the goal is reached and backtracks when a space cannot be connected.

Figure 3.6(d) is particularly interesting because it demonstrates our use of $\mathcal{C}_R^{free}$ connectivity. As opposed to the local planner approach employed in PLR [25], MANIP-SEARCH does not directly attempt to connect two neighboring points in $\mathcal{C}_R$. MANIP-SEARCH searches all actions in the manipulation space to join the configuration space components occupied by the robot and the subgoal. The procedure finds that it is easiest to pull the couch from one side and then go around the table for access. This decision resembles human reasoning and cannot be reached with existing navigation planners.

Figure 3.6(a) also demonstrates a weakness of L_1 planning. Suppose the couch was further constrained by the table such that there was no way to move it. Although the table is obstructing the couch, the table does not explicitly disconnect any free-space and would therefore not be considered for motion.

Figure 3.7 is a more complex example with backtracking. In the lower frame, we changed the initial configuration of the table. The initial call to RCH still plans through

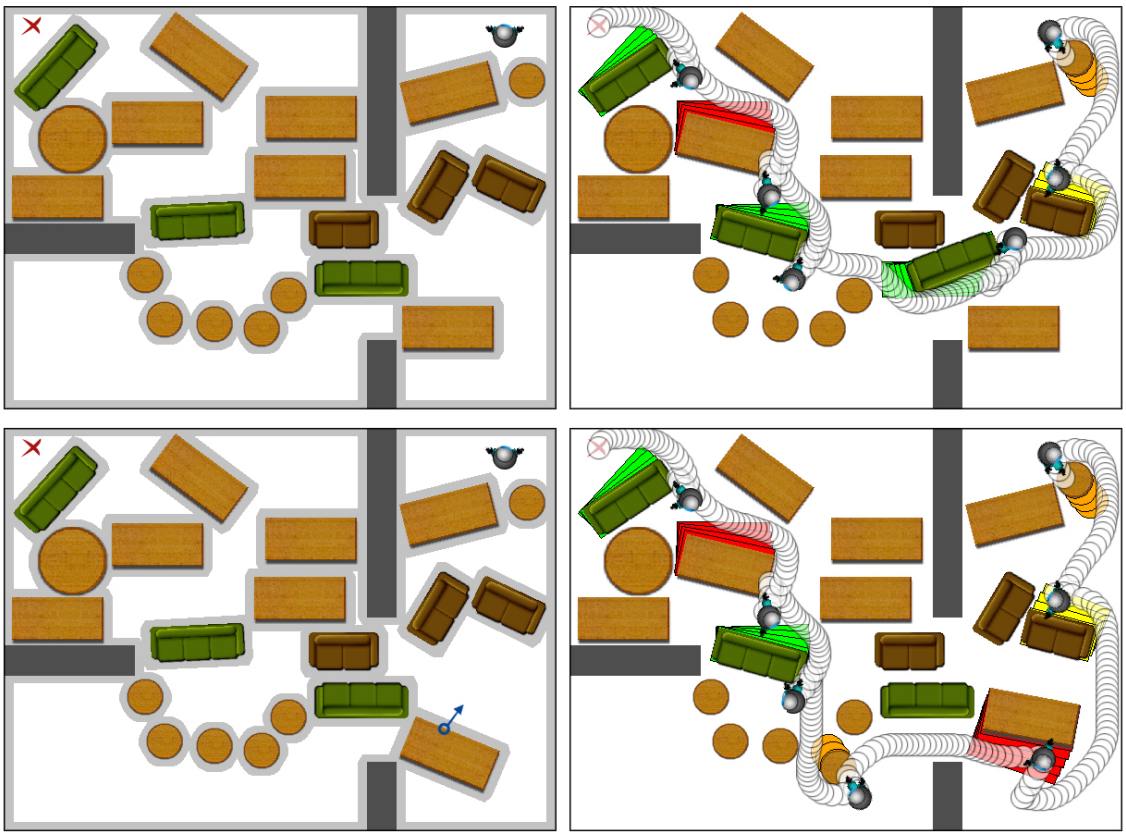


Figure 3.7: The generated plan output by our dynamic simulation NAMO planner is illustrated by the time-lapse sequences on the right.

the couch, however MANIP-SEARCH finds that it cannot be moved. The planner backtracks, calling RCH again and selects an alternative route.

Figures 3.7 and 3.6.3 show the scalability of our algorithm to problems with more movable objects. While computation time for Figure 3.6(a) is $< 1s$, the solutions for 3.7 and 3.6.3 were found in 6.5 and $9s$ respectively (on a Pentium 4 3GHz). Notice that the planning time depends primarily on the number of manipulation plans that need to be generated for a solution. Although the largest example contains 90 movable obstacles, compared to 20 in Figure 3.7, there is no sizable increase in the solution time.

Finally, consider the simple examples for which BFS examined tens of thousands of states in Figure 2.1. The solution to (a) is found instantly by the first heuristic search

Figure 3.8: A larger scale example consisting of 90 movable obstacles. Two separate plans are computed and demonstrated in our dynamic simulation.

after examining 15 states. Both (b) and (c) are solved after considering 147 states. This number includes states considered during heuristic search, *Navigate* path search and the verification of connectivity at each step of *Manipulate*. (d) is solved after 190 states.

3.6.4 Analysis

SelectConnect has clear advantages over ConnectFS in terms of both average computation time and ease of implementation. Implemented as best first search, SelectConnect is not globally optimal. Note, however, that for each choice of obstacle, the planner still selects a motion with least cost. If planning efficiency and space are not the

primary concern, uniform cost or A^* variants would restore optimality. We now prove L_1 completeness for SELECTCONNECT.

Lemma 3.6.4. *Any solution found by* SC$(W^t, PrevList, r^f)$ *is valid in NAMO.*

Proof. This can be seen from the construction of the algorithm. By induction: in the base case SELECTCONNECT returns a single valid $Navigate(W^t, \tau_N(r^t, r^f))$ on line 3. Such a path exists by definition of $\mathcal{C}_R^{acc}(W^t)$ (3.3.1).

Assume that the call to SELECTCONNECT$(W^{t+2}, PrevList, r^f)$ on line 11 returns a valid *FuturePlan*. SC$(W^t, \ldots)$ pre-pends *FuturePlan* with $Navigate(W^t, \tau_N(r^t, r^{t+1}))$ and $Manipulate(W^{t+1}, O_l, G_i, \tau_M(r^{t+1}, r^{t+2}))$ operators. τ_M is valid due to the completeness of MANIP-SEARCH (3.5.2) and τ_N is valid since $r^{t+1} = \tau_M[0] \in \mathcal{C}_R^{AC}(W^t)$ by construction of MANIP-SEARCH and Def. 3.3.4. Hence, SC$(W^t, PrevList, r^f)$ also returns a valid plan. By induction every plan returned by SELECTCONNECT is valid. □

Lemma 3.6.5. *Suppose there exists a* 1-solution *to* $K(W^t, C_F)$ *which displaces* O_F *and* $C_F \notin PrevList$. *Then the call to* SELECTCONNECT$(W^t, PrevList, r^f)$ *will either find a valid solution to* $NAMO(W^t, r^f)$ *or call* MANIP-SEARCH(W^t, O_F, C_F).

Proof. Since there exists a *1-solution* to $K(W^t, C_F)$, Lemma 3.5.2 ensures that there exist $r_F \in C_F$ and $\tau_1(r^t, r_F)$ such that for all s, $\tau_1[s] \in \bigcap_k \overline{\mathcal{X}_R(F_k)} \bigcap_{j \neq F} \overline{\mathcal{X}_R^{O_j}(W^T)}$. Let τ_2 be any path in $\mathcal{C}_R$ from r_F to r^f. Let τ_F be the compound path (τ_1, τ_2).

On the first call to RCH$(W^t, AvoidList, PrevList, r^f)$ (line 4), *AvoidList* is empty. We are given that $C_F \notin PrevList$. Let $r_m \in C_F$ be the first state in τ_1 that is in C_F. Such a state must exist since $r_F \in C_F$. Since all τ_F states, r_i $(i < m)$, cannot be in any obstacle other than O_F, they are either in $\mathcal{C}_R^{free}$ or exclusively in O_F, satisfying the conditions of Def. 3.6.1. Hence, τ_F is $Valid(O_F, C_F)$. Since a $Valid(O_F, C_F)$ path exists, RCH will find a path (Lemma 3.6.2).

The loop on lines 4-12 will terminate only if SC succeeds in solving NAMO on line 8 or RCH fails to find a path on line 4. Line 12 of RCH ensures that on each iteration the pair (O_j, C_j) returned by RCH is distinct from any in *AvoidList*. This pair is added to

AvoidList. Since there are finite combinations of obstacles and free space components, the loop must terminate. However, τ_R will remain $Valid(O_F, C_F)$ and RCH will find paths until it returns (O_F, C_F). Therefore either a NAMO solution will be found or RCH will return (O_F, C_F). In the latter case, line 6 of SC calls MANIP-SEARCH(W^t, O_F, C_F). □

Theorem 3.6.6. SELECTCONNECT *is resolution complete for problems in* L_1.

Proof. Let NAMO(W^0, r^f) be an L_1 problem. We will show that SELECTCONNECT(W^0, r^f) finds a solution. In the base case, $x^f \in \mathcal{C}_R^{acc}(W^t)$ and line 3 yields the simple *Navigate* plan. In the following, let $\Omega = \{C_1, \ldots, C_n\}$ be an ordered set of disjoint free space components that satisfies Def. 3.5.1 for the given problem.

Assume SELECTCONNECT(W_{i-1}, r^f) has been called such that W_{i-1} is a world state resulting from a sequence of *1-solutions* to $K(W_{j-1}, C_j)|C_j \in \Omega, j < i)$. By definition of L_1 there exists a *1-solution* to $K(W_{i-1}, C_i)$ that moves some obstacle O_F (3.5.1). From Lemma 3.6.5 we have that SC(W_{i-1}, r^f) will either find a sequence of valid actions that solve NAMO(W_{i-1}, r^f) or call MANIP-SEARCH(W_{i-1}, O_F, C_i) on line 6. Since MANIP-SEARCH is resolution complete it will return a solution (τ_M, W^i, c) where W_i is the next state in the sequence of solutions to $K(W_{j-1}, C_j)|C_j \in \Omega, j \leq i)$. SELECTCONNECT$(W_{i-1}, r^f)$ will call SELECTCONNECT(W_i, r^f).

By induction, if SELECTCONNECT does not find another solution it will find the solution indicated by Ω. Each recursive call to SC adds a C_i to *PrevList*. When all C_i are added to *PrevList*, there are no $Valid(O_F, C_i)$ paths and RCH will return NIL (Lemma 3.6.3). Hence the maximum depth of recursion is the number of C_i. Analogously, each loop in lines 4-12 adds a distinct pair (C_i, C_j) to *AvoidList* such that when all pairs are added RCH will return NIL. Hence the maximum number of calls made from each loop is the number of such pairs. Therefore the algorithm will terminate. □

3.7 Summary

In this chapter we made initial progress towards planning for Navigation Among Movable Obstacles. First, we gave a configuration space representation for NAMO problems. By

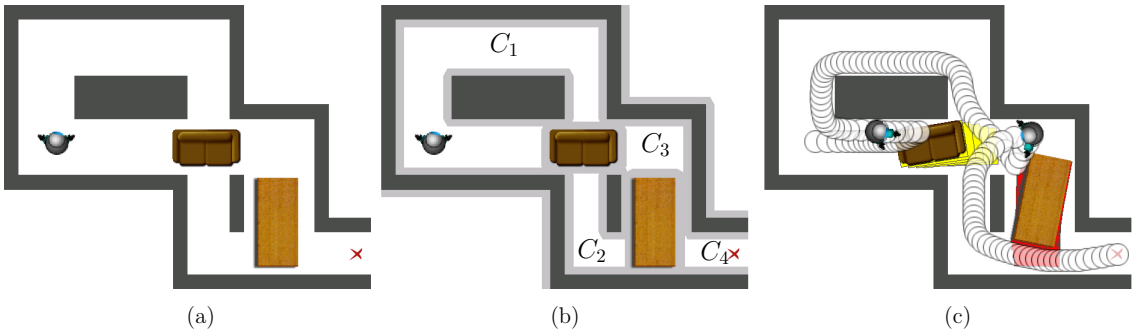


Figure 3.9: SELECTCONNECT solves the open problem considered difficult by Chen.

analyzing the relationship between action spaces for *Navigate* and *Manipulate* operators we identified useful goals for these actions. This analysis gave us tools for constructing a conceptual planner and a practical implementation that solves problems in the intuitive L_1 class. Notably, the complexity of search was reduced from the number of objects in the scene to the difficulty of the *Navigation* task. In fact, we solved problems with nearly 100 movable obstacles in seconds.

In addition to handling problems of high dimensionality, SELECTCONNECT plan is also effective in domains with complex geometry. In their analysis, Chen [25] presented one difficult puzzle problem shown in Figure 3.7. The PLR planner pushes the love seat into C_3 and cannot recover. Using $\mathcal{C}_R^{free}$ connectivity, SELECTCONNECT considers connecting C_3 as one of the options and successfully solves the example.

Clearly, there are many problems that do not fall into the L_1 class. These problems require us to consider cases where moving one object affects the robot's ability to move another. The algorithms presented in this chapter do not consider these possibilities. However, we will show that our representations and methods for reasoning about spatial interactions are easily extended to more complex domains. So far, we have focused on the interaction between *Navigate* and *Manipulate* operators. Chapter 6 develops the relationship between *Manipulate* operators acting on different objects and analyzes the effect they have on each other's domains. First, Chapters 4 and 5 will show how decisions to move objects are applied in autonomous execution.

4

Humanoid Manipulation

So far our investigation of NAMO has been largely theoretical. We showed that it is possible to decide the movement strategy for a robot that can manipulate obstacles in a large cluttered environment. In this chapter we will address the control problem of executing the desired motion on a humanoid robot. The robot used in our experiments is the Kawada HRP-2. This robot has the capacity for navigation and manipulation of large objects. Its anthropomorphic kinematics make it suitable for interacting in human environments. Furthermore, implementation on HRP-2 allowed us to study the interaction between navigation and manipulation from the perspective of multi-objective control.

We are primarily interested in manipulation of large objects such as carts, tables, doors and construction materials. Small objects can be lifted by the robot and modeled as additional robot links. Heavy objects are typically supported against gravity by external sources such as carts, door hinges or construction cranes. Yet, neither wheeled objects nor suspended objects are reliable sources of support for the robot. Large, heavy objects are interesting because they require the robot to handle significant forces while maintaining balance. We present a method that generates trajectories for the robot's torso, hands and feet that result in dynamically stable walking in the presence of known external forces.

In NAMO, the robot interacts with unspecified objects. Consequently the interaction forces are rarely known. While small variations can be removed by impedance control

and online trajectory modification, larger correlated errors must be taken into account during trajectory generation. To account for this, we give a method for learning dynamic models of objects and applying them to trajectory generation. By using learned models we show that even 55kg objects, equal to the robot's mass, can be moved along specified trajectories.

4.1 Related Work

Early results in humanoid manipulation considered balance due to robot dynamics. Inoue [42] changed posture and stance for increased manipulability and Kuffner [43] found collision-free motions that also satisfied balance constraints. These methods did not take into account object dynamics. In contrast, Harada [44] extended the ZMP balance criterion for pushing on an object with known dynamics. Harada [45] also proposed an impedance control strategy for pushing objects during the double support phase of walking. We focus on continuous manipulation during all walking phases.

With the introduction of preview control by Kajita[46], Takubo [47] applied this method to adapting step positioning while pushing on an object. Nishiwaki[48][49] proposed that the external forces from pushing could be handled by rapid trajectory regeneration. Yoshida [50][51] locally modified the planned path for a light carried object to avoid collisions introduced by applying preview control. Our work extends beyond pushing and modification to realizing a desired trajectory for a heavy object. Furthermore, in contrast to assuming that objects are known or external sources of error, we learn about their response to our force inputs.

Recently, most studies of interaction with unknown objects have been kinematic. Krotkov[52] and Fitzpatrick[53] studied impulsive manipulation to detect the affordances of various objects through different sensing modalities. Stoychev[54] considered learning to use objects for specific behaviors and Christiansen[55] learned to manipulate an object between a set of discrete states. However, for large objects the controller must account for the continuous dynamic effect they have on balance and stability.

While our focus is on learning the dynamic model of an unknown *object*, this paper is closely related to modeling *robot* dynamics. Atkeson [56] summarizes approaches to learning or adapting parameters to achieve precise trajectory following. Friction modeling, in particular has been studied extensively as summarized by Canudas [57] and Olsson[58]. More sophisticated methods for learning the dynamics of tasks in high dimensional spaces are studied by Atkeson, Moore and Schaal [59][60].

4.2 Biped Control with External Forces

Biped locomotion keeps the robot upright by pushing on the ground with the robot's legs. To generate any desired vertical forces the stance foot must be in contact with the ground. Let the center of pressure or Zero Moment Point (ZMP) be the point about which the torques resulting from all internal and external forces acting on the robot sum to zero. A necessary condition for maintaining ground contact is that the ZMP be within the area of the stance foot [61, 62]. If the ZMP leaves the foot, the robot tips about a foot edge.

The most common method for maintaining the position of the ZMP is by generating and following trajectories for the robot torso. This method accomplishes two goals. First, it achieves the desired displacement of the torso and satisfies any kinematic constraints. Second, in ensures a desired position for the ZMP throughout the duration of trajectory execution and therefore implies that the vertical forces necessary for supporting the robot can be effected continuously. In our case, trajectories are re-computed at $.15s$ intervals.

This section details the control strategy for walking manipulation that takes into account external forces. The controller consists of three significant elements:

- Decoupling the object and robot trajectories.
- Trajectory generation satisfying ZMP and object motion.
- Online feedback for balance and compliance.

Instantiating these three components lifts control from the 30 dimensional space of robot joints to a higher level of abstraction: a system that realizes a single object trajectory.

4.2.1 Decoupled Positioning

At the highest level, we represent the manipulation task as a system of two bodies. The object, **o**, and robot, **r**, are attached by horizontal prismatic joints to a grounded stance foot. The stance foot position changes in discrete steps at a constant rate $k = 900ms$. Section 4.2.2 computes independent workspace trajectories for $\mathbf{x}_r$ and $\mathbf{x}_o$. To implement this abstraction we describe how workspace trajectories map to joint space.

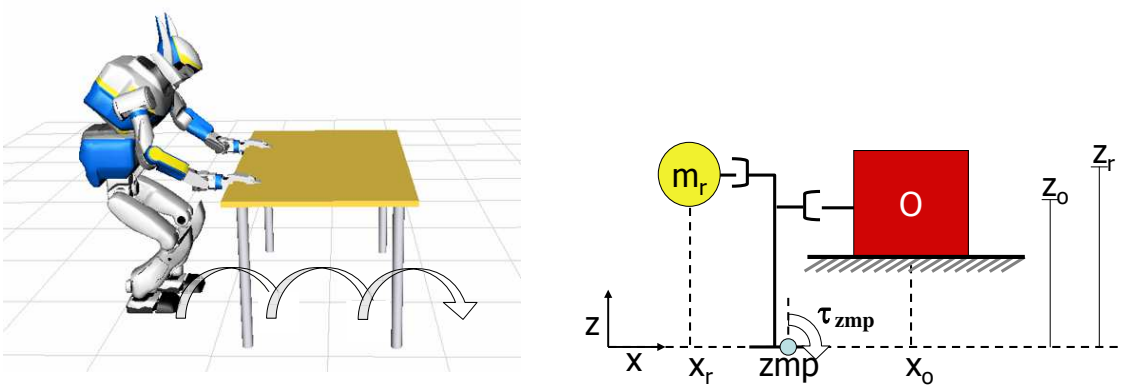
We start the mapping by defining the trajectories for hands and feet relative to the object. Due to rigid grasp manipulation, the hand positions, $\mathbf{p}_{lh}$ and $\mathbf{p}_{lr}$ remain at their initial displacements from $\mathbf{x}_o$. For simpler analysis, the stance foot $\mathbf{p}_{st}$ is fixed relative to $\mathbf{x}_o$ at each impact. The robot swing foot, $\mathbf{p}_{sw}$ follows a cubic spline connecting its prior and future stance positions. To achieve a fixed displacement from the object on each step, the object velocity is bounded by the maximum stride length and step rate. We restrict the values of $\dot{x}_o$ in advance.

We also fix the trajectory for the robot torso, $\mathbf{p}_{torso}$ relative to $\mathbf{x}_r$. Although the center of mass position, $\mathbf{x}_r$, is a function of all the robot links we assume that $\mathbf{x}_r$ remains fixed to $\mathbf{p}_{torso}$ after grasp. This assumption is relaxed in Section 4.2.2 through iterative controller optimization. Notice that *although many of the link positions are highly coupled, the two positions of interest* $\mathbf{x}_r$ *and* $\mathbf{x}_o$ *are not.*

Suppose we have workspace trajectories for both $\mathbf{x}_o$ and $\mathbf{x}_r$. The former specifies trajectories for hands and feet and the latter defines $\mathbf{x}_{torso}$. Joint values that position the four ungrounded links are found with resolved rate control [63].

$\mathbf{p}_{st}$	$\rightarrow$	$\mathbf{x}_r$	6 Stance leg	$\mathbf{x}_r$	$\rightarrow$	$\mathbf{p}_{lh}$	7 L arm
$\mathbf{x}_r$	$\rightarrow$	$\mathbf{p}_{sw}$	6 Swing leg	$\mathbf{x}_r$	$\rightarrow$	$\mathbf{p}_{rh}$	7 R arm

These solutions complete the mapping from any valid workspace placement of $\mathbf{x}_r$ and $\mathbf{x}_o$ to robot joints. We compared analytical inverse kinematics (IK) to a gradient method based on the pseudo-inverse of the robot Jacobian. Analytical IK allowed faster computation, avoided drift and assured that the solutions would satisfy joint limit constraints. The two



(a) Geometric Model (b) Computational Model

Figure 4.1: Model of the robot and object used in our work.

chest joint values were constants that maximize the workspace. Redundancy in the arms is resolved by fixing elbow rotation about the line connecting the wrist and shoulder.

4.2.2 Trajectory Generation

Section 4.2.1 gave a mapping from commanded workspace positions of $\mathbf{x}_r$ and $\mathbf{x}_o$ to joint positions. We now focus on workspace control. Given a commanded trajectory for $\mathbf{x}_o$ we compute a trajectory for $\mathbf{x}_r$ that satisfies balance constraints.

We define balance by relating the zero moment point to stance foot position. Let x be the direction of object motion. z_o is the height of the hands and f is the reflected force. Eq. 4.2.1 introduces zmp as the ground point around which the torques due to gravity acting on $\mathbf{x}_r$, reflected force from accelerating $\mathbf{x}_r$ and from the object sum to zero.

$$\tau_{zmp} = m_r g(x_r - zmp) - m_r \ddot{x}_r z_r - z_o f = 0 \tag{4.2.1}$$

Solving for zmp yields:

$$zmp = x_r - \ddot{x}_r \frac{z_r}{g} - \frac{z_o f}{m_r g}. \tag{4.2.2}$$

Dynamic balance requires zmp to remain in the robot support polygon. To maximize

error tolerance we seek a trajectory that minimizes the distance between zmp and the stance foot center $zmp_d = x_{st}$. *Recall that x_{st}, and thus zmp_d are known given a trajectory for $\mathbf{x}_o$.* (S.4.2.1)

Let $J_0 = \sum_t (zmp^t - zmp_d^t)^2$ be the performance index for balance. Eq. 4.2.3 further defines β and β_d as functions of zmp_d and x_r respectively.

$$\beta_d = zmp_d + \frac{z_o f}{m_r g} \quad \beta = x_r - \ddot{x}_r \frac{z_r}{g} \tag{4.2.3}$$

Substitution yields $J_0 = \sum_t (\beta^t - \beta_d^t)^2$. Notice that zmp_d is the trajectory of foot centers and $\{z_o, m_r, g\}$ are constants. Hence, assuming that f is known, the trajectory of future values for β_d is fully determined.

Suppose we interpret β as the observation of a simple linear system in x_r with the input $\dddot{x}_r$. For smoothness, we add squared input change to the performance index.

$$J = \sum_{t=1}^{\infty} Q_e(\beta^t - \beta_d^t)^2 + R(\dddot{x}^t - \dddot{x}^{t-1})^2 \tag{4.2.4}$$

We can now determine the optimal $\dddot{x}_r$ with preview control [46]. At any time t we know the error $e(t) = \beta^t - \beta_d^t$, state $\mathbf{x}(t) = [\, x_r^t \;\dot{x}_r^t \;\ddot{x}_r^t \,]^T$ and N future β_d^i. Appendix B gives the procedure for pre-computing the gains G_1, G_2 and G_3 such that the incremental control in Eq. 4.2.5 minimizes J.

$$\Delta \dddot{x}_r^t = -G_1 e(t) - G_2 \Delta x_r^t - \sum_{i=1}^{N} G_3^i (\beta_d^{t+i} - \beta_d^{t+i-1}) \tag{4.2.5}$$

The control $\Delta \dddot{x}_r$ is discretely integrated to generate the trajectory $\{\ddot{x}_r, \dot{x}_r$ and $x_r\}$ for $\mathbf{x}_r$. The trajectory for y_r is found by direct application of preview control since the object reflects no forces tangent to x.

Since $\mathbf{x}_r$ is assumed to be fixed to the robot torso, the generated joint space trajectory still results in zmp tracking error. We incorporate this error into the reference trajectory and iterate optimization.

4.2.3 Online Feedback

Section 4.2.2 described the generation of a balanced trajectory for $\mathbf{x}_r$ given $\mathbf{x}_o$. To handle online errors we modify these trajectories online prior to realization with robot joints. Online feedback operates at a $1ms$ cycle rate.

Accumulated ZMP tracking error can lead to instability over the course of execution. Therefore, a proportional controller modifies the acceleration of $\mathbf{x}_r$ to compensate for ZMP errors perceived through the force sensors at the feet. These corrections are discretely integrated to achieve $\mathbf{x}_r$ position.

The trajectory for $\mathbf{x}_o$, or the robot hands, is modified by impedance. We use a discrete implementation of the virtual dynamic system in Eq. 4.2.6 to compute the offset for $\mathbf{x}_o$ that results from integrating the measured force error F.

$$F = m_i\ddot{\mathbf{x}}_o + d_i\dot{\mathbf{x}}_o + k_i(\mathbf{x}_o - \mathbf{x}_o^d) \tag{4.2.6}$$

Impedance serves two goals. First of all, we ensure that hand positioning errors do not lead to large forces pushing down on the object. Since the robot does not use the object for support, d_i and k_i are set low for the z direction.

Second, we prevent the robot from exceeding torque limits when the trajectory cannot be executed due to un-modeled dynamics. The position gain for the x direction trades a displacement of $10cm$ for a $100N$ steady state force. This allows for precise trajectory following and soft termination when the trajectory offset exceeds force limits.

4.3 Modeling Object Dynamics

Section 4.2 described our implementation of whole body manipulation given a known external force. However, when the humanoid interacts with an unspecified object, the reflected forces may not be known in advance. A reactive strategy for handling external forces might assume that the force experienced when acting on the object will remain constant for .15 seconds, or the duration of trajectory execution. In this section we

present an alternative method that improves performance by learning the mapping from a manipulation trajectory to the reflected object forces.

4.3.1 Motivation for Learning Models

Modeling addresses two challenges: noise in force sensor readings and the dependence of balance control on future information. The former is common to many robot systems. While high frequency forces have no significant impact on balance, low frequency force response must be compensated. The complication is that online filtering introduces a time delay of up to $500ms$ for noise free data. Modeling can be used to estimate forces without time delay.

For balancing robots such as humanoids, we not only require estimates of current state but also of future forces. Typically, a balance criterion such as center of pressure location (ZMP) is achieved by commanding a smooth trajectory for the robot COM. [46] demonstrates that accurate positioning of ZMP requires up to two seconds of future information about its placement. Since external forces at the hands create torques that affect the ZMP, they should be taken into account two seconds earlier, during trajectory generation. Hence, the purpose of modeling is to use known information such as the target object trajectory to accurately predict its low frequency force response in advance. The predicted response is used to generate a smooth trajectory for the robot COM that satisfies the desired ZMP.

4.3.2 Modeling Method

Environment objects can exhibit various kinematics and dynamics including compliance, mechanical structure and different forms of friction. For instance, the tables and chairs used in our experiments are on casters. Each caster has two joints for wheel orientation and motion. Depending on the initial orientation of the wheels the object may exhibit different dynamics. Currently, we do not have a perception system that can detect and interpret this level of modeling detail. Consequently we approach this problem from the

perspective of finding a simple and effective modeling strategy.

First, we observe that despite the complex kinematic structure of a humanoid robot, the robot is typically modeled as a point mass attached to the stance foot with prismatic joints. Likewise, an object can be modeled as a point mass in Eq. 4.3.1. Given experimental data we could compute the mass and friction for an object and use them to predict force. However, due to uncertainty in caster orientation and the low velocities of manipulation our experiments showed that even this model was unnecessarily complex. We did not find a consistent relationship between acceleration and force. Consequently we chose to base our model solely on viscous friction as shown in Eq. 4.3.2.

$$f^t = m_o\,\ddot{x}_o^t + c\,\dot{x}_o^t. \tag{4.3.1}$$

$$f^t = c\,\dot{x}_o^t \tag{4.3.2}$$

To find $\mathbf{c}$ that satisfies this relationship we applied least squares regression on collected data. We executed a trajectory that displaced the object at distinct velocities, $\dot{x}_o^t$, and measured the force at HRP-2s hands, f^t, at millisecond intervals. The collected data was represented in Eq. 4.3.3. The term b was used to remove bias which appeared as a constant force offset allowed by impedance control after grasp.

$$\begin{bmatrix} \dot{x}^1 & \dot{x}^2 & \cdots & \dot{x}^n \\ 1 & 1 & \cdots & 1 \end{bmatrix}^T \begin{bmatrix} c \\ b \end{bmatrix} = \begin{bmatrix} f^1 & f^2 & \cdots & f^n \end{bmatrix}^T \tag{4.3.3}$$

The solution to this set of over-constrained equations is found simply by applying the right pseudo-inverse. During data collection we applied a reactive balancing strategy which assumed a constant force response during a $.15s$ trajectory cycle. This approach was sufficiently stable for brief interactions.

4.4 Experiments and Results

We conducted experiments on model-based whole body manipulation using a loaded table on casters, as shown in Figure 4.2. The robot grasped the table and followed a smooth

(a) t=0s (b) t=20s (c) t=0s

(d) t=20s (e) t=28s (f) t=38s

Figure 4.2: HRP-2 pushes then pulls 30 and 55kg loaded tables.

trajectory for $\mathbf{x}_o$ as generated from a joystick input. Our results were compared to a rigid grasp implementation of a reactive approach to handling external forces presented in [49], which assumed that sensed forces would remain constant. Both methods recomputed the trajectory for $\mathbf{x}_r$ every $.15s$, at which time the reactive strategy updated its estimated force. The reactive method was applied first to gather data and learn an object model from Section 4.3.2. Brief experiments of less than $10s$ we necessary to collect the data. We applied both methods on a series of experiments that included a change of load such that the total mass ranged from $30kg$ to $55kg$.

4.4.1 Prediction Accuracy

First, we look at how well our model predicts force. The comparisons in this subsection use data from experiments that are not used to build the model. The comparison in Table 4.4.1 shows that the mean squared error between modeled and measured force is

	MSE $F_{err}(N)$		MSE $x_{err}^{PC}(m)$	
	model	react	model	react
$30kg$	4.44	9.19	.427	.674
$55kg$	5.21	12.5	.523	.971

Table 4.1: Prediction Accuracy

	ZMP SD (m)		Force SD (F)	
	model	react	model	react
$30kg$	.0214	.0312	11.06	15.79
$55kg$	.0231	.0312	12.15	46.25

Table 4.2: System Stability

lower than the error of assuming that force remains constant during the control cycle.

Since preview control takes into account future β_d, including predicted force, next we propose a more accurate prediction measure. Let β_d reflect the difference in predicted and actual force. Preview control is applied to find a trajectory that compensates for the simulated error. It generates an erroneous x_r displacement, x_{err}^{PC}, during the $150ms$ that the trajectory is active. x_{err}^{PC} is the expected trajectory error given the error in force prediction.

The comparison between the expected trajectory error, shown in Figure 4.4 and Table 4.4.1, also favors the model based method. x_{err}^{PC} decreases if we assume a faster control cycle. However, even for a $20ms$ cycle, we found that error decreases proportionally for both controllers and the ratio of their MSE remains in favor of modeling.

4.4.2 System Stability

The accuracy of prediction has significant effect on the overall stability of the controlled system. Incorrect predictions affect the trajectories for $\mathbf{x}_r$ and $\mathbf{x}_o$. First consider the resulting ZMP of the robot. While both controllers exhibit a slight offset in ZMP from grasping the object, the constant error can be removed with integral or adaptive control. A greater concern is the variance in this error. Figure 4.5 shows the increased noise in the control signal for ZMP when using the reactive control. Table 4.4.1 summarizes this effect.

An even clearer distinction between the two methods is directly reflected in the noise of the perceived force data. Table 4.4.1 also shows the variance in the noise given the off-line filtered signal. The difference in noise is clearly seen in Figure 4.3.

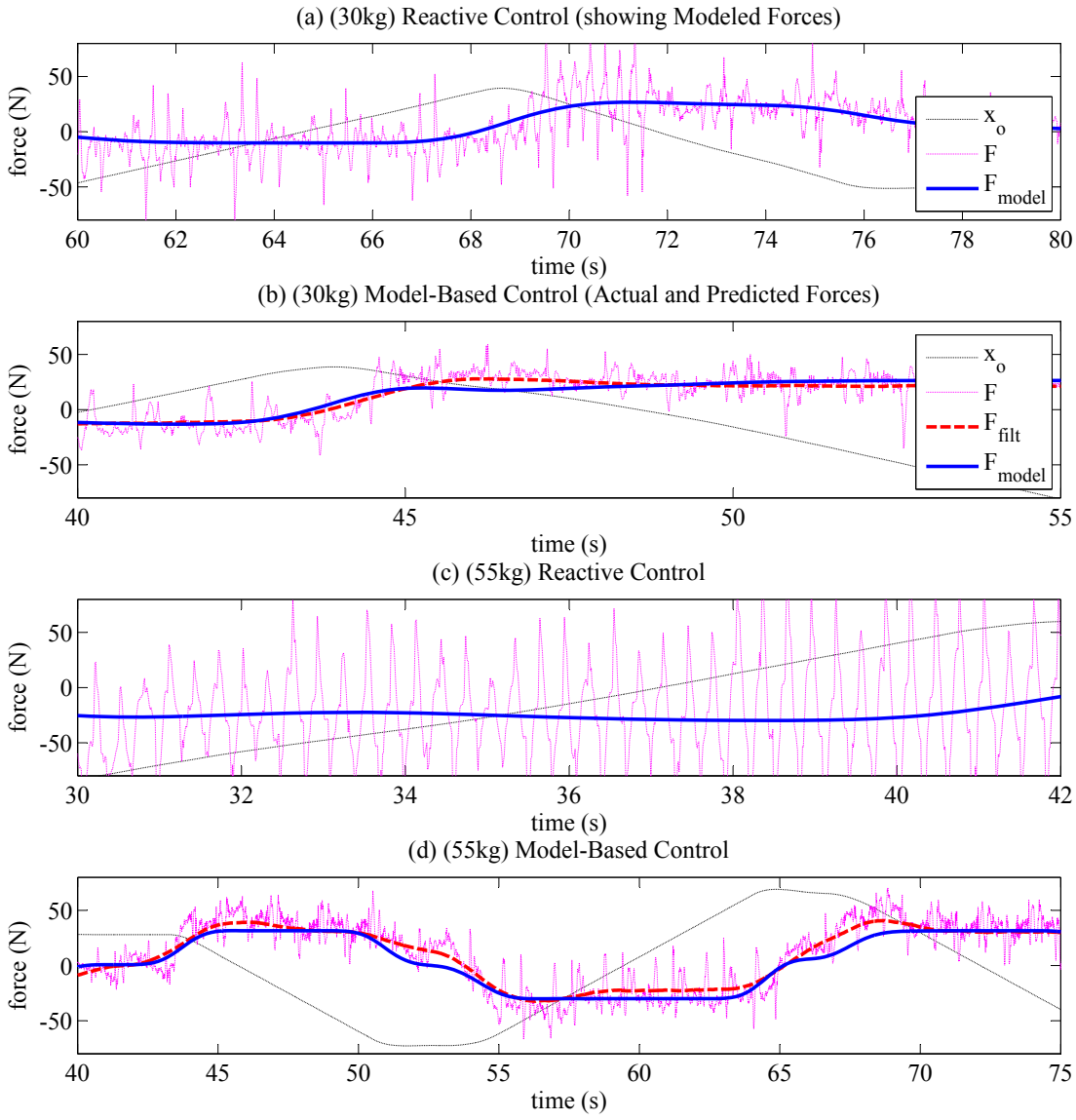


Figure 4.3: Comparison of forces experienced with reactive and model-based control.

4.4.3 Discussion

Our experiments show a significant improvement both in prediction accuracy and system stability when using the learned object model for control. One of the most convincing results is the accurate force prediction for a $55kg$ object in Figure 4.3(d). Additionally, notice the low noise variance in sensed forces when using the model based controller.

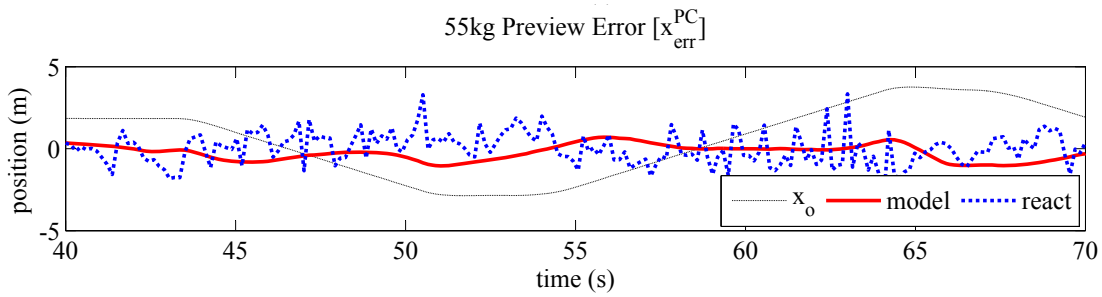


Figure 4.4: Trajectory error introduced by preview control with erroneous prediction.

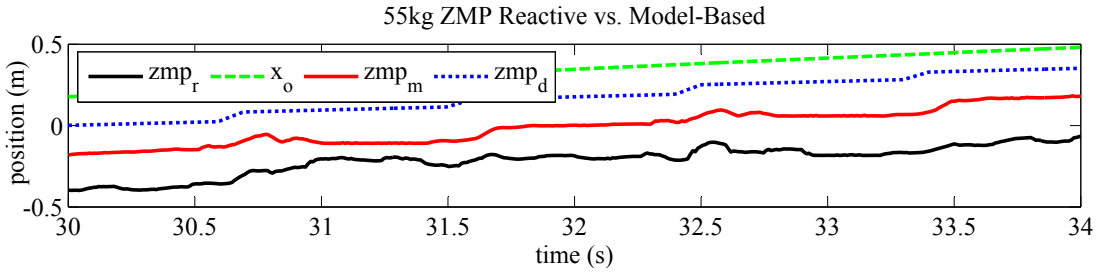


Figure 4.5: Realized ZMP for identical reference trajectories

We also evaluated the performance of the system with an inaccurate model. Due to online balance compensation and impedance it was possible to manipulate the $55kg$ table using the $30kg$ feed-forward model. However, as seen in Figure 4.6, the impedance controller saturates at the object force and the ZMP error error grows considerably.

4.5 Summary

In this chapter we have shown that it is possible to reliably manipulate unknown, large, heavy objects such as tables along specified trajectories with existing humanoid robots. We found that statistical methods such as least squares regression can be used to learn a dynamic model for the unknown object and use it to improve balance during manipulation. Our estimated model of viscous friction was compared to a reactive method which assumed that forces would remain constant. Learning models proved to yield better prediction of forces and increase balance stability when used during whole body manipulation.

In future work, we are interested in merging the feed-forward model with feedback

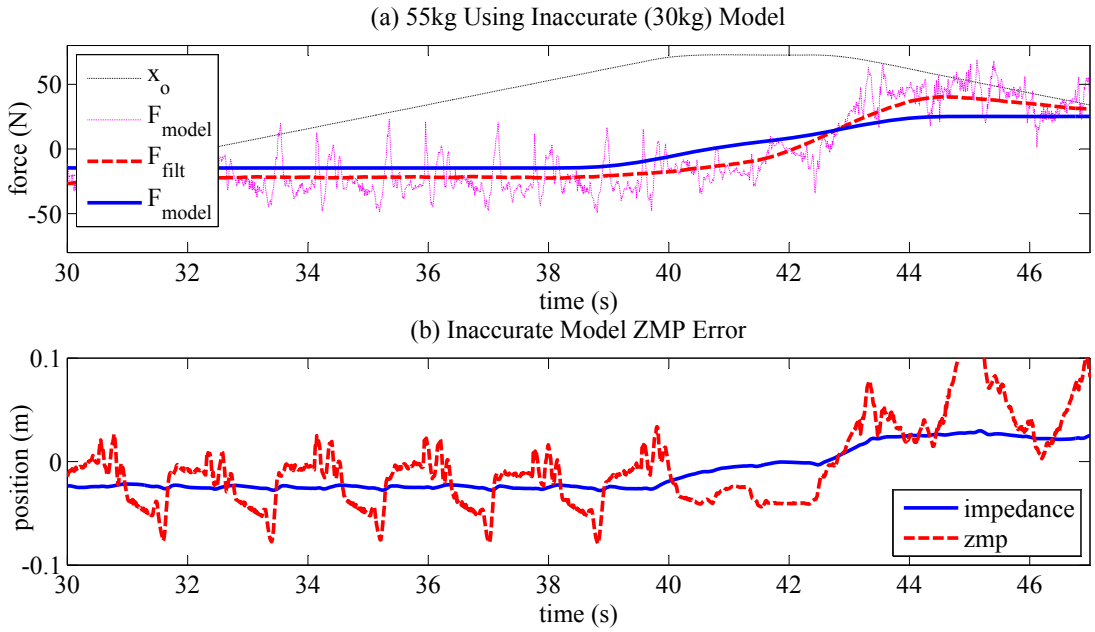


Figure 4.6: Manipulation of a 55kg table with 30kg model.

information. State estimators such as Kalman filters could be used to maximize the performance of the robot by combining information sources. Furthermore, adaptive control could be used to handle online changes in friction and mass.

While our experiments were restricted to rigid grasp manipulation for objects on casters, similar techniques can be applied to very different scenarios. Two important classes are objects that can be lifted by the robot and objects that are suspended by cranes rather than carts. In the case of the former, the robot can statically estimate object mass by measuring the load after lifting. The latter cannot be lifted and will require a similar experimental trajectory execution. For suspended objects inertial terms will likely dominate friction. In this case, we propose estimation of mass and inertia.

We have now presented methods for planning object motion and controlling a robot to perform the desired movement. Chapter 5 will detail our complete architecture for NAMO execution.

5

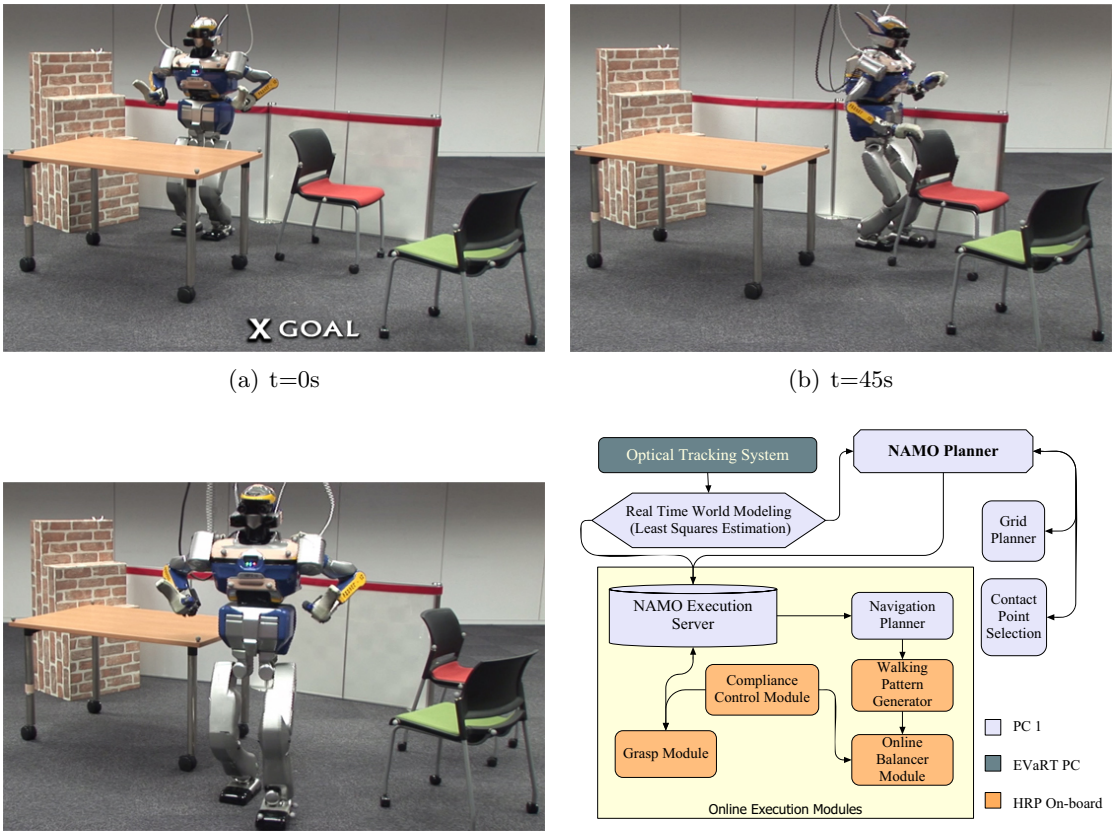
System Integration

5.1 From Planning to Execution

Having investigated a strategy for NAMO planning and a method for mobile manipulation by humanoid robots we now turn our attention to merging these elements into a complete system for Navigation Among Movable Obstacles. This chapter introduces the architecture for our implementation of NAMO using the humanoid robot HRP-2 shown in Figure 5.1. We present the methods used for measuring the environment, mapping world objects into a planar search space and constructing motion plans for robots. The NAMO planner used in this chapter is derived from Chapter 3. The details of control for robot walking and whole body manipulation were addressed in Chapter 4.

The architecture in this chapter is distinct from previous humanoid systems planners which focus on specified tasks such as navigation and manipulation. Sakagami, Chestnutt and Gutmann and plan navigation using stereo vision.[64, 65, 66]. Kuffner, Harada, Takubo, Yoshida and Nishiwaki generate dynamically stable trajectories for manipulation given an environment model. [44, 67, 49, 47]. Brooks recognizes doors and defines an opening behavior [68]. Existing systems do not allow the robot to perceive the environment and perform arbitrary manipulation. Our domain requires autonomous localization, planning and control for walking, grasping and object manipulation.

Our implementation was performed in a $25m^2$ office setting consisting of tables and



(a) t=0s

(b) t=45s

(c) t=180s

(d) NAMO Architecture

Figure 5.1: Autonomous execution of NAMO with architecture diagram.

chairs. All movable objects are on casters to simplify the task of manipulation. This domain is representative of hospitals, homes and nursing homes where heavy non-wheeled objects are typically stationary. Furthermore, our instrumentation approach to measurement is feasible in these environments.

5.2 Measurement

In order to apply NAMO planning the robot must first acquire a geometric model of its environment. Our entire system gathers information from three three sources: real-time external optical tracking, joint encoders and four six-axis force sensors. The force sensors at the feet and hands are discussed in Chapter 6 with regard to closed loop control. In this

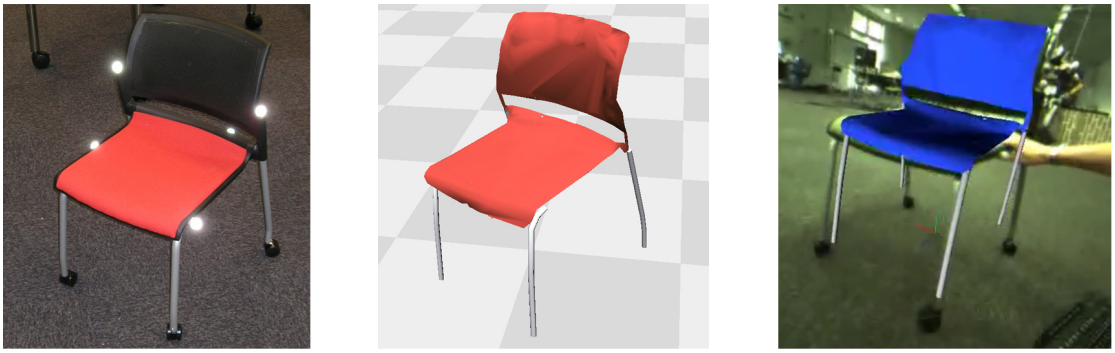
section we focus on external optical tracking for the robot and movable objects. Individual robot links are positioned by combining tracking for the robot torso with encoder readings for joint angles.

The most common method for recognizing and localizing indoor objects is visual registration of features perceived by an onboard camera. Approaches such as the Lucas-Kanade tracker [69] are summarized by Haralick and Forsyth [70, 71]. While these methods are portable, Dorfmuller points out that the speed and accuracy of a tracking system can be enhanced with hybrid tracking by the use of markers [72]. In particular, he advises the use of retro-reflective markers. Some systems use LED markers [73], while others combine vision-based approaches with magnetic trackers [74]. Given our focus on planning and control, we chose an accurate method of perception by combining off-line geometric modeling with online localization.

5.2.1 Object Mesh Modeling

The robot world model consists of the robot and two types of objects: movable and static. Static objects cannot be repositioned and must always be avoided. Limited interaction with static objects prompted us to use approximate bounding box models to represent their geometry. Movable objects require manipulation during which the robot must come close to the object and execute a precise grasp. These objects were represented internally by 3D triangular mesh models.

Our experimental environment contained two types of movable objects (chairs and tables). To construct accurate models of these objects we used the Minolta Vivid laser scanner. The resulting meshes shown in Figure 5.2(b) were edited for holes and processed to minimize the overall polygon count while ensuring that at least one vertex exists in every $0.125m^3$ voxel of the mesh. This simplified object detection in any given region of 3D space to vertex inclusion.



(a) Chair and markers (b) Mesh model (c) Real-time Overlay

Figure 5.2: Off-line modeling and online localization of a chair.

5.2.2 Recognition and Localization

Precise object localization and model fitting was achieved using the EVa Real-Time Software (EVaRT) with the Eagle Motion Analysis optical tracking system. Each model was assigned a unique arrangement of retro-reflective markers. Under the assumption that the object is a rigid body, any six dimensional configuration of the object corresponds to a unique set of configurations for the markers. We define a *template* as the set of x,y and z coordinates of each marker in a reference configuration.

EVaRT continuously tracks the locations of all the markers to a height of 2 meters. Distances between markers are calculated to .3% accuracy leading to a position error of less than $1mm$. When tracking approximately 60 markers the acquisition rate is 60Hz. Matching the distances between markers, EVaRT partitioned them among objects and provided our system with sets of marker locations and identities for each object. The detected markers are rigidly transformed from the template and permit a linear relationship in the form of a transformation matrix.

Since some markers can be occluded from camera view, we add redundant markers to the objects and perform pose estimation using only the visible set. Estimation is a two step procedure given in Figure 5.2.2. At this time, we do not enforce rigidity constraints. Even for a system with only four markers, the accuracy of marker tracking yields negligible

Given positions for unoccluded template, $\{\mathbf{a}_1, ..., \mathbf{a}_n\}$, and localized markers $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$:

1. Find the centroids ($\mathbf{c}_a$ and $\mathbf{c}_b$) of the template markers points and the observed marker locations. Estimate the translational offset $\hat{\mathbf{t}} = \mathbf{c}_b - \mathbf{c}_a$. Removing this offset, $\mathbf{b}'_i = \mathbf{b}_i - \hat{\mathbf{t}}$ places the markers at a common origin.

2. Next we define a linear system for the orientation of the object,

$$\begin{bmatrix} \mathbf{a}_1^T & & \mathbf{0} \\ & \mathbf{a}_1^T & \\ \mathbf{0} & \cdots & \mathbf{a}_1^T \\ \mathbf{a}_n^T & & \mathbf{0} \\ & \mathbf{a}_n^T & \\ \mathbf{0} & & \mathbf{a}_n^T \end{bmatrix} \begin{bmatrix} \hat{\mathbf{r}}_1 \\ \hat{\mathbf{r}}_2 \\ \hat{\mathbf{r}}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{b}'_1 \\ \mathbf{b}'_2 \\ \vdots \\ \mathbf{b}'_n \end{bmatrix} \qquad \text{such that} \qquad \mathbf{b}_i = \begin{bmatrix} - & \hat{\mathbf{r}}_1^T & - & \\ - & \hat{\mathbf{r}}_2^T & - & \hat{\mathbf{t}} \\ - & \hat{\mathbf{r}}_3^T & - & \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{a}_i$$

   expresses an estimate of the object transform. We solve this system for $\hat{\mathcal{R}}$ online using LQ decomposition.

Figure 5.3: Object localization procedure.

shear and scaling in the estimated transformation. The transform is optimal in minimizing the summed squared 3D marker error.

We refer to the collection of transformed meshes for the movable objects, static objects and the robot as the *world model*. Poses for individual robot links are found by combining joint encoder readings with the tracked position and orientation of the robot torso. The entire model is continuously updated at 30hz. Further details and applications of our approach to mixed reality experimentation can be found in [75].

5.3 Planning

The world model, constructed in Section 5.2.2, combined with kinematic and geometric models of the robot is sufficient to implement NAMO planning. However, the search space for such a planner would have 38 degrees of freedom for robot motion alone. The size of this space requires additional considerations which are taken into account in Chapter 8. Presently, we observe that for many navigation tasks the two dimensional subspace consisting of the walking surface is sufficiently expressive. In fact, larger objects such as

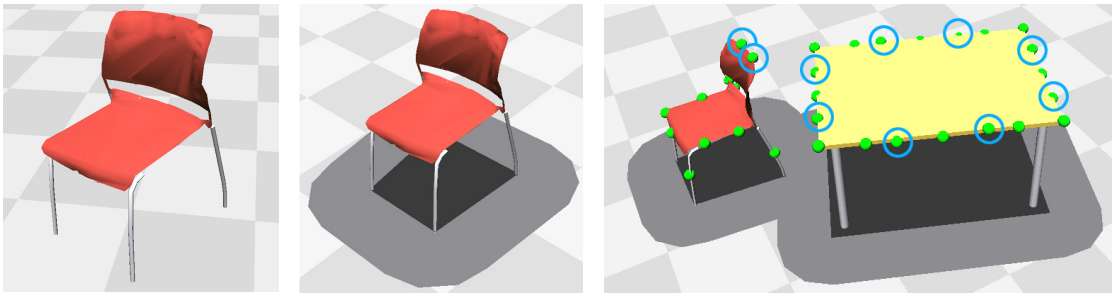
tables and chairs that inhibit the robot's path must be pushed rather than lifted [44, 67]. Hence, their configurations are also restricted to a planar manifold. Reducing the search space to two dimensions makes it possible to apply our NAMO implementations from Chapters 3 and 4 directly to real environments. To do so, we map the world model to a planar configuration space. We also introduce a definition for contact and an abstract action space for the robot.

5.3.1 Configuration Space

Mapping the world model into a configuration space that coincides with our previous NAMO implementations requires us to project all objects onto the ground plane. Each object is associated with the planar convex hull of the projected mesh vertices. Figure 5.4 shows the model of a chair projected onto the ground plane. While our algorithms are general for any spatial representation, the projected space is computationally advantageous since it considers only three degrees of freedom for the robot and objects. Currently, the space does not allow interpenetration between objects. Alternative implementations could use multiple planes to allow penetration at distinct heights.

The robot is represented by a vertical cylinder centered at the robot torso. A $0.3m$ radius safely encloses torso motion. The cylinder projects to a circle on the ground plane. We pre-compute the navigational configuration space, $\mathcal{C}_R$ of the robot. Each $\mathcal{C}_R$ obstacle (O_i) is a Minkowski sum of the robot bounds with the corresponding planar object. For navigation, the robot can be treated as a point that moves through the $\mathcal{C}$-space. Lighter shaded ground regions around projected obstacles in Figure 5.4(b) are $\mathcal{C}_R$ obstacles. The robot may walk on the lighter regions but its centroid must not enter them.

To further decrease computation costs, we used the Bentley-Faust-Preparata (BFP) approximate convex hull algorithm to compute $\mathcal{C}_R$ obstacles [76]. The resulting obstacles have significantly fewer vertices and edges while subject to only 1% error. Decreasing the number of edges reduces the cost of testing for robot collision. The error is acceptable since we can adjust the radius of robot safety bounds.



(a) Mesh model (b) Projection (c) Contact Selection

Figure 5.4: Mapping objects into the planning domain.

5.3.2 Contact Selection

As part of NAMO planning, the robot must select locations for contacting movable objects. Our implementation uses rigid grasp manipulation and automatically selects points for grasps. In our work, we have found three essential criteria for this selection:

1. *Proximity to object perimeter* - Ensure that the point is in the robot's workspace when standing next to the object.

2. *Restricted quantity* - Limit the number of possible interactions to a discrete set of grasp points to increase the efficiency of planning.

3. *Uniform dispersion* - Provide contact points for any proximal robot configuration.

We introduce one solution for locating such points. One can interpret the convex hull from the previous section as a representation of the object perimeter. Starting at an arbitrary vertex of this hull, our algorithm moves along the edges and places reference points at equidistant intervals. The interval is a parameter that we set to $0.2m$.

For each reference point, we find the closest vertex by Euclidian distance in the full 3D object mesh along the horizonal model axes. The selected vertices are restricted to lie in the vertical range of $[0.5m, 1.0m]$. When interpreted as contact points, we have found that these vertices satisfy the desired criteria for objects such as tables and chairs. Selected points can be seen in Figure 5.4(c). The robot successfully performed power

grasps of our objects at the computed locations.

More complex objects may be widest outside the operating range or may have geometry that restricts the locations and orientations of valid grasps. These cases can be handled by applying a grasp planner such as GraspIt to determine valid contacts. [27] The proposed criteria can still be optimized by using proximity to the contour points as a heuristic for selecting grasps.

5.3.3 Action Spaces

So far we have given definitions for the geometry of the configuration space and for allowable contacts. The NAMO planner also requires us to define the action space of the robot that will be searched in selecting *Navigate* and *Manipulate* operators.

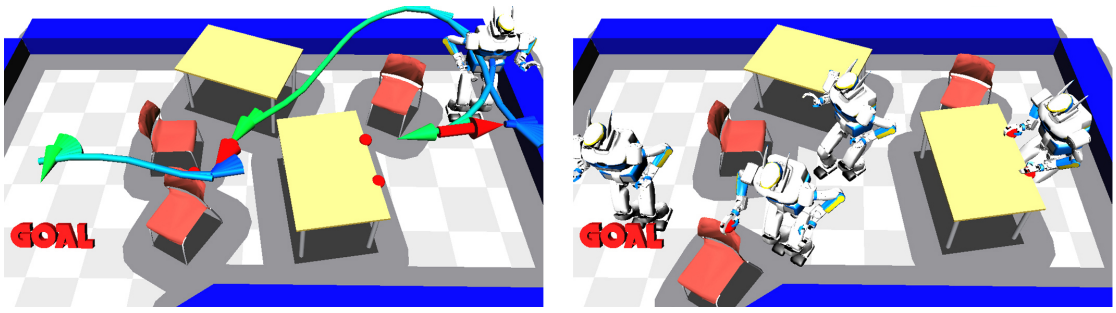
Humanoid walking has the property of being inherently discrete since a stable motion should not terminate in mid-stride. Each footstep is a displacement of the robot. One option for planning is to associate the robot base with the stance foot and directly plan the locations of footsteps [77, 65]. This creates discrete jumps in the location of the robot base at each change of supporting foot. In our approach, we define an abstract *base* and plan its motion along continuous paths. The simplest mapping of the base trajectory to footsteps places feet at fixed transforms with respect to the base on every step cycle. We found that a horizontal offset of $9cm$ from the base along a line orthogonal to the base trajectory yields repeatable, safe and predictable robot motion.

The *Navigate* space consists of base displacements. Since the foot trajectory generated from base motion must be realizable by the controller, we restricted base motions to ones that translate into feasible footstep gaits. From any base configuration, the robot is permitted 41 discrete actions which are tested for collisions with the $\mathcal{C}$-space obstacles:

1. One $0.1m$ translation backward.

2. Twenty rotations in place in the range of 30^o.

3. Twenty $0.2m$ translations forward with a rotation in the range of 30^o.



(a) Initial state and NAMO plan (b) Traces of execution

Figure 5.5: Simulated example shows NAMO plan and traces of execution.

During search, the planner keeps track of visited states. A* finds least cost paths and terminates when all states have been visited.

Having grasped an object *Manipulate* searches the same space as *Navigate*. During planning we assume that the object remains at a fixed transform with respect to the robot base and therefore the stance foot by construction. Our planner distinguishes large objects, such as tables, from smaller objects such as chairs. Smaller objects have less inertia and can be manipulated safely with one arm. Tables, however, have a significant impact on the robot dynamics and are grasped with two hands. While both grasps are restricted to the computed contact points, there are fewer contact configurations for the robot base during a two handed grasp.

Action definitions complete the description of a NAMO problem and allow us to apply the planner to solve problems such as the one in Figure 5.5. In this case we applied SelectConnect. The figure shows light arrows to indicate planned navigation and dark arrows for manipulation. The robot joint configurations shown in Figure 5.5(b) interpret planned actions as described in Chapter 4.

5.4 Uncertainty

The NAMO planner presented in Chapter 3 assumes that the world conforms to the actions determined by the planner. While precise modeling and localization of objects

serves to decrease error, significant uncertainty remains during execution. Planning with uncertainty or in partially known environments are complex problems. Erdmann corners the system into a desired state, [19]. Stentz efficiently replans while expanding knowledge of the environment [78]. Kaelbling finds optimally successful plans [20]. One or more of these approaches can be adapted to NAMO planning. In this thesis we present only the necessary solution to uncertainty that makes execution possible.

We consider the complete NAMO implementation as a hierarchy of high level action planning, lower level path planning and online joint control. At the lowest end of the spectrum, insignificant positioning errors can be handled by joint-level servo controllers. At the highest, a movable obstacle that cannot be moved may require us to recompute the entire NAMO plan. We propose a minimalist strategy to error recovery. Each level of the hierarchy is implemented with a strategy to reduce uncertainty. When this strategy fails, execution is terminated. Further development should consider reporting the error and considering alternatives at higher levels of the architecture. Presently, we describe three of the strategies applied in our work.

5.4.1 Impedance Control

At the lowest level of the hierarchy uncertainty in our estimate of object mass and robot positioning is handled with Impedance Control as described in Section 4.2.3. The goal of this controller is to handle small positioning errors that occur due to robot vibration and environment interaction. This level ensures that the robot does not damage itself or the obstacle during interaction by limiting the forces that the robot can exert in order to achieve the precise positioning demanded by the planner.

5.4.2 Replanning Walking Paths

Since the lowest level of execution does not guarantee precise placement of objects, it is possible that the navigation paths computed by the NAMO planner will not be possible after the displacement of some object. Furthermore, since robot trajectories are executed

open loop with regard to localization sensors, the objects and the robot may not reach their desired placements.

In order to compensate for positioning errors, the path plans for subsequent *Navigate* and *Manipulate* actions are re-computed at the termination of each NAMO action. At this time the planner updates the world model from the optical tracker and finds suitable paths for the robot. Notice that we do not change the abstract action plan with regard to object choices and orderings. We simply adapt the actions that the plan will take to ensure their success. For *Navigate* paths we iterate state estimation, planning and execution to bring the robot closer to the desired goal. This iteration acts as a low rate feedback loop from the tracker to the walking control and significantly improves the robot's positioning at the time of grasp.

5.4.3 Guarded Grasping

Although we have described the execution of *Navigate* and *Manipulate* actions as two essential components of NAMO, bridging these two actions is also an interesting problem. Having navigated to a grasp location, the robot is required to execute a power grasp of the object at a given contact point. Positioning errors in grasping can be more severe than manipulation since the object is not yet fixed to the robot and its location is uncertain.

Having walked up to an object, the robot must grasp it and then walk with it. HRP-2 reacquires the world model and determines the workspace position for its hand. It preshapes the hand to lie close to this position and then performs a guarded move to compensate for any perception error. This process is not only functionally successful but also approximates human grasping behavior as described in [79].

The initial workspace hand position for grasping is obtained by selecting a point $.05m$ above the object and $.05m$ closer to the robot (in the direction of robot motion). The robot moves its hands to this position via cubic spline interpolation from its estimated state. Subsequently the robot moves its hand downward and forward to close the $.05m$ gaps. This guarded move is executed using impedance control to prevent hard collisions

	NAMO	Navigation	Execution
Figure 5.1(a) (Real)	$0.06s$	$0.09s$	$63.0s$
Figure 5.5 (Simulated)	$0.13s$	$0.93s$	$153.0s$

Table 5.1: NAMO Implementation: Run times for Planning and Execution

and is terminated when the force sensors reach a desired threshold. We ensure that the robot's palm is in contact with the top surface of the object and the thumb is in contact with the outer edge. The remaining fingers are closed in a power grasp until the finger strain gauges exceed the desired threshold.

5.5 Results

Our complete NAMO system was successfully applied to a number of simulated examples such as the one presented in Figure 5.5 as well as the real robot control problem in Figure 5.1(a). The simulated evaluations involved all planning aspects of our work, including dynamically stable walking pattern generation for *Navigate* and *Manipulate* actions. In our laboratory experiments, the robot perceived its environment and autonomously constructed a plan for reaching the goal. After walking to approach an object, HRP-2 successfully grasped it and walked to create the planned displacement. Subsequently it was able to reach the desired goal.

Table 5.5 details the high-level NAMO planning time, replanning time for navigation and the total execution time for the two presented examples. Notice that the NAMO planning time is three orders of magnitude smaller than the actual time for executing the computed motion. This makes it feasible to view the NAMO system as a real-time generalization of modern path planning techniques to worlds where strictly collision-free paths are not available.

5.6 Summary

This section completes our initial investigation of Navigation Among Movable Obstacles for mobile manipulators. We have shown that it is feasible to plan interactions in

scenes with large numbers of objects and execute the interactions with a humanoid robot. Furthermore, we have provided a sample architecture for how such a system would be integrated with online sensing. *To our knowledge, our robot is the first to move an unspecified object out of its way and perform a desired task.*

At present our implementation handles uncertainty in action through compliance and replanning of paths. These methods are suitable for domains with low modeling error that does not change the overall NAMO plan. In domains with greater model uncertainty the robot may need to respond to unexpected collisions as well as change its decisions about which objects to manipulate. Further integration of NAMO planning and execution should prove to be of great interest in constructing autonomous behaviors for service robots.

The following chapters will describe extensions to NAMO from the perspective of motion planning. We will consider interactions between objects that block each other's motion, articulated robots that operate in full 3D spaces and constrained objects such as doors and drawers. The motion strategies determined in all of these domains can be executed by systems similar to the one we have described.

6

Object Interaction

6.1 Introduction

Solving Navigation Among Movable Obstacles requires the robot to reason about how manipulating objects will change its capabilities. In Chapter 3 our focus was specifically on the robot's ability to *Navigate* or find paths from an initial configuration to a goal. We presented an algorithm that determines which objects should be moved and a strategy for increasing the robot's free space. Now, we observe that navigation is not the only capability of the robot that can be constrained by obstacles. Like a desired navigation path, a desired displacement of an object may also be impossible due to the presence of other objects. In this chapter we present an algorithm that manipulates objects to create space for additional manipulation as well as robot navigation.

In contrast to domains like assembly planning where objects can be removed from the scene, NAMO planners are faced with the practical challenge of being constrained by the geometry of the environment [14]. Consider navigating to the goal in Figure 6.1. This problem has the appearance of an L_1 domain since there is only one table between the robot and the goal. However, the robot must first move three more objects to make space for the table in order to solve the problem. Our algorithm autonomously generates the solution shown in the figure.

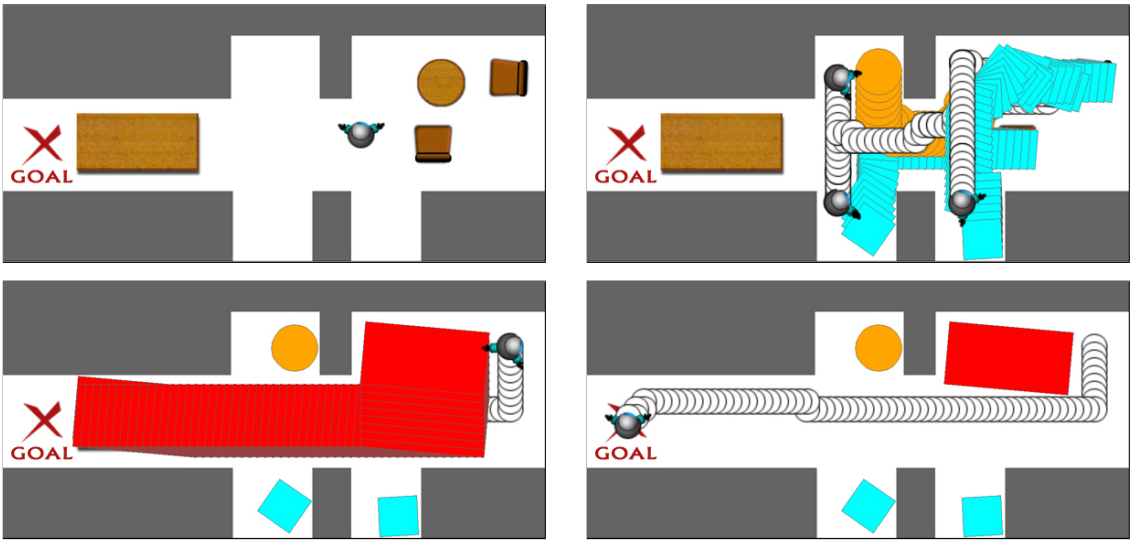


Figure 6.1: Simulated solution requires the robot to move four objects.

To better understand the challenge of handling interactions between manipulated objects, consider extending SELECTCONNECT to L_k problems where up to k objects may be moved to connect free space components. This is non-trivial even when $k = 2$, or the problem requires us to move two objects. In the best case, every robot path between two components would pass through two objects, O_1 and O_2, allowing the planner to locally search the joint motion space of size $|O_1| \times |O_2|$. However, as seen in Figure 6.1, the path between C_i, $C_j \in \mathcal{C}_R^{free}$ might only pass through one object (the table). A complete L_2 planner must then consider all possibilities for the choice of second object. In general, for L_k problems, a single keyhole will require the enumeration of 2^{k-1} possible sets of objects that do not directly interfere with a path to the goal.

In this chapter we propose to use *artificial constraints* to manage this complexity. Our approach does not escape the curse of dimensionality. Instead, we give a procedure for fast heuristic search in domains where standard proximity heuristics provide little or no insight. First, we extend the classification of NAMO problems to include restrictions on the number of times each object is displaced. We show that even *monotone problems* that require one displacement per object retain a very complex search space. Our algorithm

assumes that the domain is monotone and uses the assumption to constrain the search space. An alternative scheme proposed by Nieuwenhuisen uses randomized search and increases the probability of displacing colliding objects [80]. Our constraint approach is related to [81]. Rather than assuming a priority on object motions we search the space of object choices and orders.

6.2 Extended Hierarchy

In order to manage the increased complexity when local search requires the motion of multiple objects, we propose further classification of the movable obstacle domain to *monotone* plans. In assembly planning, monotone plans refer to plans where each application of an operator yields a subassembly that is part of the final assembly [14]. We do not enforce a final assembly and define *monotone plans* as those in which a Manipulated object cannot be moved again:

$$W^{t+1} = Manipulate(W^t, O_i, G_i, \tau) \quad \Rightarrow \quad q_i^T = \tau_{O_i}(1) \quad (T > t) \tag{6.2.1}$$

Monotone search decouples the joint motion space of objects into individual path plans. The search decides which objects to displace, a path for each object and the ordering of object motion. Notice that any plan can be expressed as a sequence of monotone plans:

$$Plan_{NM} = \ldots, \tau_1, (O_i, \tau_2), \tau_3, (O_j, \tau_4), \tau_5, (O_i, \tau_6), \tau_7, \ldots \equiv$$
$$Plan_{M_1} = \ldots, \tau_1, (O_i, \tau_2), \tau_3, (O_j, \tau_4), \tau_5 \text{ and } Plan_{M_2} = (O_i, \tau_6), \tau_7, \ldots \tag{6.2.2}$$

Let W^6 be the world state after the operation $Navigate(W^5, \tau_5)$, prior to the second displacement of O_i. We refer to W^6 as an intermediate world state. A problem can be characterized in its *non-monotone degree* by the number of intermediate states necessary to construct a sequence of monotone plans. We propose the following classes of problems:

L_k Linear problems where components of $\mathcal{C}_R^{free}$ can be connected independently. k is the maximum number of objects that must be displaced to connect two components.

NL Non-linear problems require the planner to consider interactions between keyholes.

M Monotone problems where each object only needs to be moved once.

NM_i Non-monotone problems that can be expressed as i monotone problems with intermediate states.

Planners operate in one or two of these classes. For instance an L_3NM_6 planner would seek linear solutions that require manipulating at most three objects and using six intermediate states to merge two free space components. Our proposed algorithm operates in L_kM.

6.3 Planning in Monotone Domains

The monotone class of problems helps organize the study of movable objects. It also preserves a number of the computational challenges that arise from interdependent *Manipulate* operators. A monotone planner determines a subset $\{O_1, \ldots, O'_m\} = \mathbf{O}'_M \subset \mathbf{O}_M$ of movable objects to displace. It also finds a valid set of paths $\{\tau_{O_1}, \ldots, \tau_{O'_m}\}$ for displacing the objects and $\{\tau_1, \ldots, \tau_{m+1}\}$ *Navigate* operations between grasps. Additionally, the planner decides an ordering for object motion. This section will analyze the problem complexity and present our solution.

6.3.1 Forward Search

Suppose we were to perform a standard forward search of obstacle motion. In the monotone case, we do not need to consider all possible *Navigate* and *Manipulate* paths. At each time-step t we select an object O_i for motion and a goal configuration $q_i^{t+2} \in \mathcal{C}_{O_i}^{place}(W^{t+2})$. We verify that there exists an accessible contact configuration $r^{t+1} \in \mathcal{C}_R^{AC}(O_i, W^t) \subset IK(\mathbf{G}(O_i), W^t)$. We also check the existence of valid paths:

$$\begin{aligned} &Navigate(W^t, \tau_1(r^t, r^{t+1})) \text{ and} \\ &Manipulate(W^{t+1}, O_i, G_i, \tau_2(r^{t+1}, r^{t+2})) \end{aligned} \tag{6.3.1}$$

Assume that verification could be performed in constant time, and that the number of placements is $O(d^n)$, where d is the resolution of each of the n dimensions of $\mathcal{C}_{O_i}$. Typically, $n = 3$ or 6. At $t = 0$, this algorithm would select from m objects and d^n configurations for each object: $O(md^n)$. Expanding the search to depth 2, there are now $m-1$ objects and d^n placements for each object: $O(md^n \times (m-1)d^n)$. This algorithm has an asymptotic runtime of $O(md^n \times (m-1)d^n \times ... \times 2d^n \times d^n) = O(m!d^{nm})$.

The exponential runtime of this algorithm might be reduced for typical problems given a sufficiently informed heuristic. However, as seen in Figure 6.1 no notion of proximity to the goal expresses the utility of object displacement. A *useful* placement for the object is one that respects the motion of subsequent obstacles. Since the motion of future objects is postponed in the search, good placements are unknown.

6.3.2 Reverse Search

Given that the utility of an action choice depends on future action choices it appears that forward search may not be a practical method for solving L_k problems. A direct method for using information about future choices would reverse the order of action selection. Having chosen the last action, the planner uses the information from this choice in making decisions about earlier actions. Reverse planning is common in *assembly* problems. However, the implementation and motivation of reverse planning is different in our domain. Assembly planners have fixed goal configurations for all objects in which the motion of the objects is typically highly constrained. Consequently, the reverse search space has a much smaller branching factor due to *actual constraints*.

In NAMO, the final configuration is not pre-determined. In fact, reverse search has a larger branching factor than forward search since different object paths, not just placements, constitute different actions (Section 6.4). Instead of considering all possible paths, we assume that a path sample is representative of future motion. We then use the monotone assumption to create *artificial constraints* on the paths taken prior to the future motion. Currently, our approach does not consider different paths for objects. We

do, however, take into account all possible orderings in which the objects are moved.

Section 6.4 gives an explicit representation for the constraints formed by assuming future displacements in a monotone domain. Section 6.5 uses this representation in an algorithm for solving NAMO problems.

6.4 Artificial Constraints

This section defines the precise constraint that decisions about future actions place on prior motion in a monotone domain. Let W^0 be the initial world state. Assume that at some future time step $t > 0$, the robot will execute $Navigate(W^t, \tau^t(r^t, r^{t+1}))$. Let q_j^t be the configuration of obstacle O_j at time t. By the definition of free-space (Eq. 3.2.3):

$$\tau^t(s) \quad \notin \quad \mathcal{X}_R(O_j(q_j^t)) \tag{6.4.1}$$

Due to the symmetry of $\mathcal{X}$ (Eq. 3.2.2), we can invert this relationship.

$$q_j^t \quad \notin \quad \mathcal{X}_{O_j}(R(\tau^t(s))) \qquad \forall s \in [0,1] \tag{6.4.2}$$

The robot motion along τ^t defines a *swept volume* in $\mathbb{R}^n$. Let $V(\tau^t)$ be the volume of points occupied by the robot during its traversal of τ^t:

$$Navigate(W^t, \tau^t(r^t, r^{t+1})) \rightarrow V(\tau^t) \quad = \quad \bigcup_{s\in[0,1]} R(\tau^t(s)) \tag{6.4.3}$$

$$q_j^t \quad \notin \quad \mathcal{X}_{O_j}(V(\tau^t)) \tag{6.4.4}$$

Analogously, for valid $Manipulate(W^t, O_i, G_i, \tau(r^t, r^{t+1}))$, we define $V(\tau^t, O_i, G_i)$ as the volume of points occupied by the robot and the object during their joint motion:

$$Manipulate(W^t, O_i, G_i, \tau^t(r^t, r^{t+1})) \rightarrow V(\tau^t, O_i, G_i) = \bigcup_{s\in[0,1]} [R(\tau^t(s)) \cup O_i(\tau_{O_i}^t(s))] \tag{6.4.5}$$

For this motion to be valid, Eq. 3.2.5 indicates that:

$$\tau^t(s) \notin \mathcal{X}_R(O_j(q_j^t)) \qquad \tau_{O_i}^t(s) \notin \mathcal{X}_{O_i}(O_j(q_j^t))) \qquad (j \neq i) \tag{6.4.6}$$

Due to the symmetry of $\mathcal{X}$:

$$q_j^t \notin \mathcal{X}_{O_j}[R(\tau^t(s)) \cup O_i(\tau_{O_i}^t(s))] \qquad \forall s \in [0,1] \tag{6.4.7}$$

$$q_j^t \notin \mathcal{X}_{O_j}(V(\tau^t, O_i, G_i)) \tag{6.4.8}$$

Eq. 6.4.4 and 6.4.8 indicate that the swept volume of any *Navigate* or *Manipulate* operation in W^t places a constraint on the configurations of movable objects: $V^t = V(\tau^t)$ or $V(\tau^t, O_i, \mathbf{G}_{O_i}^k)$ respectively. Since objects remain fixed unless moved by *Manipulate*, then for some final time T:

$$\begin{aligned} &q_j^0 \notin \mathcal{X}_{O_j}(V^T) \text{ or there exists a time } t(0 \leq t < T) \text{ such that} \\ &Manipulate(W^t, O_j, G_i, \tau^t(r^t, r^{t+1})) \text{ and } \tau_{O_j}^t(1) \notin \mathcal{X}_{O_j}(V^T) \end{aligned} \tag{6.4.9}$$

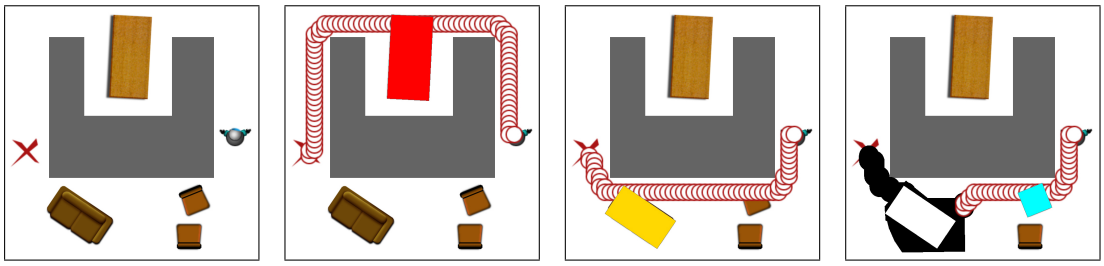
By the monotone assumption, if the initial configuration of an obstacle conflicts with V^T, there is exactly one *Manipulate* operator that displaces it to a non-conflicting configuration at some time-step t ($t < T$).

6.5 Algorithm

In order to apply the method of artificial constraints, our planner consists of two modules: *obstacle identification* and *constraint resolution*. Obstacle identification decides the last object that will be manipulated prior to reaching the goal or a subgoal. Constraint resolution plans a *Manipulate* path for this object and the following *Navigate* to the goal. The two paths form artificial constraints. We detect objects that violate the constraints in W^0 and recursively plan corresponding *Manipulate* and *Navigate* operations. The first grasping configuration identified by a successful resolution step is used as the preceding subgoal for obstacle identification. Both modules backtrack on their choices when the algorithm fails to resolve the constraints.

```
IDENTIFY-OBSTACLE(r^t, (𝕍^t, O_f, O_c))
 1  O_avoid ← ∅
 2  while O_L ← RCH_last(W^t, O_avoid) ≠ NIL
 3  do
 4      if O_L = none return Navigate(W^t, τ_N(r^t, r^f))
 5      O_c ← {O_L}
 6      (Plan, r^{t−n}, (𝕍^{t−n}, O_f^{t−n})) ← RESOLVE-CONSTRAINTS(r^t, (𝕍^t, O_f, O_c))
 7      if Plan ≠ NIL
 8        then PastPlan ← IDENTIFY-OBSTACLE(r^{t−n}, (𝕍^{t−n}, O_f^{t−n}, O_c^{t−n}))
 9             if PastPlan ≠ NIL
10               then return (PastPlan append Plan)
11      O_avoid ← O_avoid ∪ {O_L}
12  return NIL
```

Figure 6.2: Pseudo-code for IDENTIFY-OBSTACLE.



(a) Problem (b) RCH^0_{last} (Unsolved) (c) RCH^1_{last} (Solved) (d) RCH^2_{last}

Figure 6.3: RCH_{last} selects subgoals for constraint resolution.

6.5.1 Obstacle Identification

The search is initialized by a constrained relaxed planner RCH_{last}. [31] $O_L \leftarrow \text{RCH}_{last}$ operates in $\mathcal{C}_R$. It is permitted to pass through movable configuration space obstacles with a heuristic one-time cost for entering any object. RCH_{last} finds a path to the goal and selects the last colliding obstacle, O_L, to schedule for manipulation. In Figure 6.3(b), RCH^0_{last} selects the table as the last object for manipulation.

Constraint resolution, described in Section 6.5.2, validates the heuristic selection with a sequence of *Navigate* and *Manipulate* operations. If no such sequence is possible, RCH_{last} is called again, prohibiting any navigation into $\mathcal{X}_R^{O_i}(W^0)$. Since constraint reso-

lution fails on the table, RCH^1_{last} selects the couch for motion in Figure 6.3(c). We ensure completeness over the selection of final objects by aggregating $\mathbf{O}_{avoid}$, a set of prohibited navigation for RCH_{last} as described in Chapter 3.

When resolution is successful, RCH_{last} is called with the goal of reaching the initial contact configuration identified by constraint resolution. Figure 6.3(d) shows that after successfully scheduling the manipulation of the couch, RCH^2_{last} selects the chair for motion. When RCH_{last} finds a collision-free path to the subgoal, the algorithm terminates successfully.

6.5.2 Constraint Resolution

Let T index the final time step of the plan and t be the current time step. We will maintain the following sets:

$\mathbf{O}^t_f$ - the set of objects O_i scheduled for manipulation after time t.

$\mathbf{O}^t_c$ - the set of objects scheduled for motion prior to time t.

$\mathbb{V}^t$ - the union of all artificial constraints $V^{t'}(t < t' < T)$.

$\mathbb{V}^T$ is initialized as an empty volume of space. $\mathbf{O}^T_f$ and $\mathbf{O}^T_c$ are empty sets. We begin by applying $O_L \leftarrow \text{RCH}_{last}$ and adding O_L to $\mathbf{O}^T_c$. Constraint resolution attempts to move all objects from $\mathbf{O}_c$ to $\mathbf{O}_f$. Objects may be added to $\mathbf{O}_c$ when they interfere with manipulation. The following three procedures are performed recursively. Each iteration of recursion will plan from state W^t, such that operations that follow time step t are assumed to be known.

(1) Choosing an Obstacle and Contact

First, we select an obstacle $O_d \in \mathbf{O}^t_c$. We then choose a contact, G_i. Inverse kinematics yields the contact configuration r_c. If the robot is redundant the space of inverse kinematic solutions is sampled, resulting in a set of robot configurations $\{r_{c1}, r_{c2}, ..., r_{cn}\}$.

We choose a valid contact $r^{t-2} = r_{ci}$ by searching for a path $\tau_C(r_0, r^{t-2})$ which verifies that r_{ci} can be reached by the robot without passing through previously scheduled obstacles in their initial configurations:

$$\tau_g(s) \notin (\bigcup_{O_i \in \mathbf{O}_f^t} \mathcal{X}_R^{O_i}(W^0)) \cup \mathcal{X}_R^{O_d}(W^0) \qquad \forall s \in [0,1] \tag{6.5.1}$$

(2) Dual Planning for Manipulate and Navigate

The *Navigate* operation to the subsequent contact, or goal, occurs after manipulating object. Conceptually in reverse search it should be planned first. However, we have not yet determined the initial configuration for *Navigate* since it is equivalent to the final configuration of *Manipulate*. We propose assembling the *Navigate* path from two segments: τ_1 is a path from the initial contact with the object, r^{t-1}, to r^t and τ_2 is the manipulation path for the object. The robot returns to r^{t-1}, during Navigate.

τ_1) Plan a partial path τ_1 from r^{t-2} to r^t. The path must not pass through any future scheduled obstacle:

$$\tau_1(s) \notin \bigcup_{O_i \in \mathbf{O}_f^t} \mathcal{X}_R^{O_i}(W^0) \qquad \forall s \in [0,1] \tag{6.5.2}$$

Taken alone this path is not intended for a *Navigate* operation. In the world state W^{t-2}, object O_d may still block this path. We choose this path to pass through the least number of objects in their initial configuration and minimize euclidian path length. If the path is not possible, a different grasp, r^{t-2}, is selected.

τ_2) Plan $Manipulate(W^{t-2}, O_d, G_d, \tau_2^{t-2})$. The robot configuration at the start of the plan $\tau_2(0) = r^{t-2}$. The final configuration must be selected by the planner. Since

τ_2 maps to the object path τ_{2O_d}, we require the following constraints:

$$\tau_2(s) \notin \bigcup_{O_i \in \mathbf{O}_f^t} \mathcal{X}_R^{O_i}(W^0) \quad \tau_{2O_d}(s) \notin \bigcup_{O_i \in \mathbf{O}_f^t} \mathcal{X}_{O_d}^{O_i}(W^0) \quad \forall s \in [0,1] \tag{6.5.3}$$

$$\tau_{2O_d}(1) \notin \mathcal{X}_{O_d}[R(\tau_1(s)) \cup R(\tau_2(s))] \quad \forall s \in [0,1] \tag{6.5.4}$$

$$\tau_{2O_d}(1) \notin \mathcal{X}_{O_d}(\mathbb{V}^t) \tag{6.5.5}$$

Eq. 6.5.3 states that the object and the robot may not pass through the configuration space obstacles of future scheduled objects. Eq. 6.5.4 states that the final configuration of O_d may not interfere with either τ_1 or τ_2. Eq. 6.5.5 requires the final configuration of O_d to be consistent with artificial constraints.

Merging τ_1 and τ_2 into a single τ, we can define the operation $Navigate(W^{t-1}, \tau)$. The *Navigate* is valid after the obstacle has been displaced.

(3) Composing Artificial Constraints

Having selected *Manipulate* and *Navigate* operations in W^t, we can advance the search to W^{t-2}. To do so, we will update the three sets described earlier:

$$\mathbf{O}_f^{t-2} \leftarrow \mathbf{O}_f^t \cup \{O_d\} \tag{6.5.6}$$

$$\begin{aligned} \mathbb{V}^{t-2} &\leftarrow \mathbb{V}^t \cup V(\tau_1) \cup V(\tau_2, O_d, G_d) \\ &= \mathbb{V}^t \cup R(\tau_1(s)) \cup R(\tau_2(s)) \cup O_d(\tau_{2O_d}(s)) \quad \forall s \in [0,1] \end{aligned} \tag{6.5.7}$$

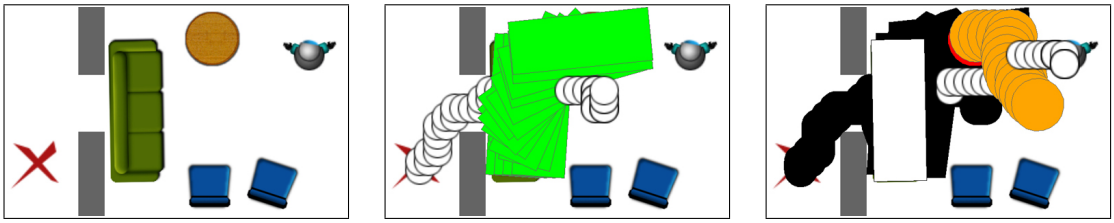
$$\mathbf{O}_c^{t-2} \leftarrow \{\, O_i \mid O_i \notin \mathbf{O}_f^{t-2} \wedge q_i^0 \in \mathcal{X}_{O_i}(\mathbb{V}^{t-2})\} \tag{6.5.8}$$

Eq. 6.5.6 fixes the configuration of O_d to the initial configuration and marks it as resolved in future states. Eq. 6.5.7 updates the artificial constraint to include the *Manipulate* and *Navigate* in W^{t-2} and W^{t-1} respectively. Eq. 6.5.8 updates the set of conflicting objects that must be moved earlier than W^{t-2} to resolve the constraints.

RESOLVE-CONSTRAINTS$(r^t, \mathbb{V}^t, \mathbf{O}_f^t, \mathbf{O}_c^t)$

if $\mathbf{O}_c^t = \emptyset$ **return** $\square$
for each $O_d \in \mathbf{O}_c$
do
  Choose $r^{t-2} : G_d(r^{t-2}) = q_d^0$
    s.t. exists $\tau_g(r_0, r^{t-2})$ **satisfying** *Eq.* 6.5.1
  Choose $\tau_1(r^{t-2}, r^t)$
    Satisfying *Eq.* 6.5.2
  Choose $\tau_2(r^{t-2}, r^{t-1})$
    Satisfying *Eq.* 6.5.3 – 6.5.5
  if no valid choices
    then return NIL
  determine $(\mathbf{O}_f^{t-2}, \mathbb{V}^{t-2}, \mathbf{O}_c^{t-2})$ **by** *Eq.* 6.5.6 – 6.5.8
  $(Plan, r^{t-n}, (\mathbb{V}^{t-n}, \mathbf{O}_f^{t-n}, \mathbf{O}_c^{t-n})) \leftarrow$
      RESOLVE-CONSTRAINTS$(r^{t-2}, (\mathbb{V}^{t-2}, \mathbf{O}_f^{t-2}, \mathbf{O}_c^{t-2}))$
  if $Plan \neq$ NIL
    then $Plan$ **append** $Manipulate(W^{t-2}, O_d, G_d, \tau_2(r^{t-2}, r^{t-1}))$
      $Plan$ **append** $Navigate(W^{t-1}, \tau_2 + \tau_1)$
      return $(Plan, W^{t-n}, (\mathbb{V}^{t-n}, \mathbf{O}_f^{t-n}, \mathbf{O}_c^{t-n}))$
return NIL

Figure 6.4: Pseudo-code for RESOLVE-CONSTRAINTS called by IDENTIFY-OBSTACLE.



(a) Problem (b) Manipulation Plan (c) Constraint Resolution

Figure 6.5: RESOLVE-CONSTRAINTS displaces table to make space for moving the couch.

6.5.3 Depth First Search

Section 6.5.1 and 6.5.2 detailed the components of our planner. Figure 6.5.1 gives the pseudo-code for IDENTIFY-OBSTACLE and Figure 6.5.2 for RESOLVE-CONSTRAINTS. The full planning algorithm is initialized by a call to IDENTIFY-OBSTACLE. It is implemented as depth first search to conserve space required for planning and help with the inter-

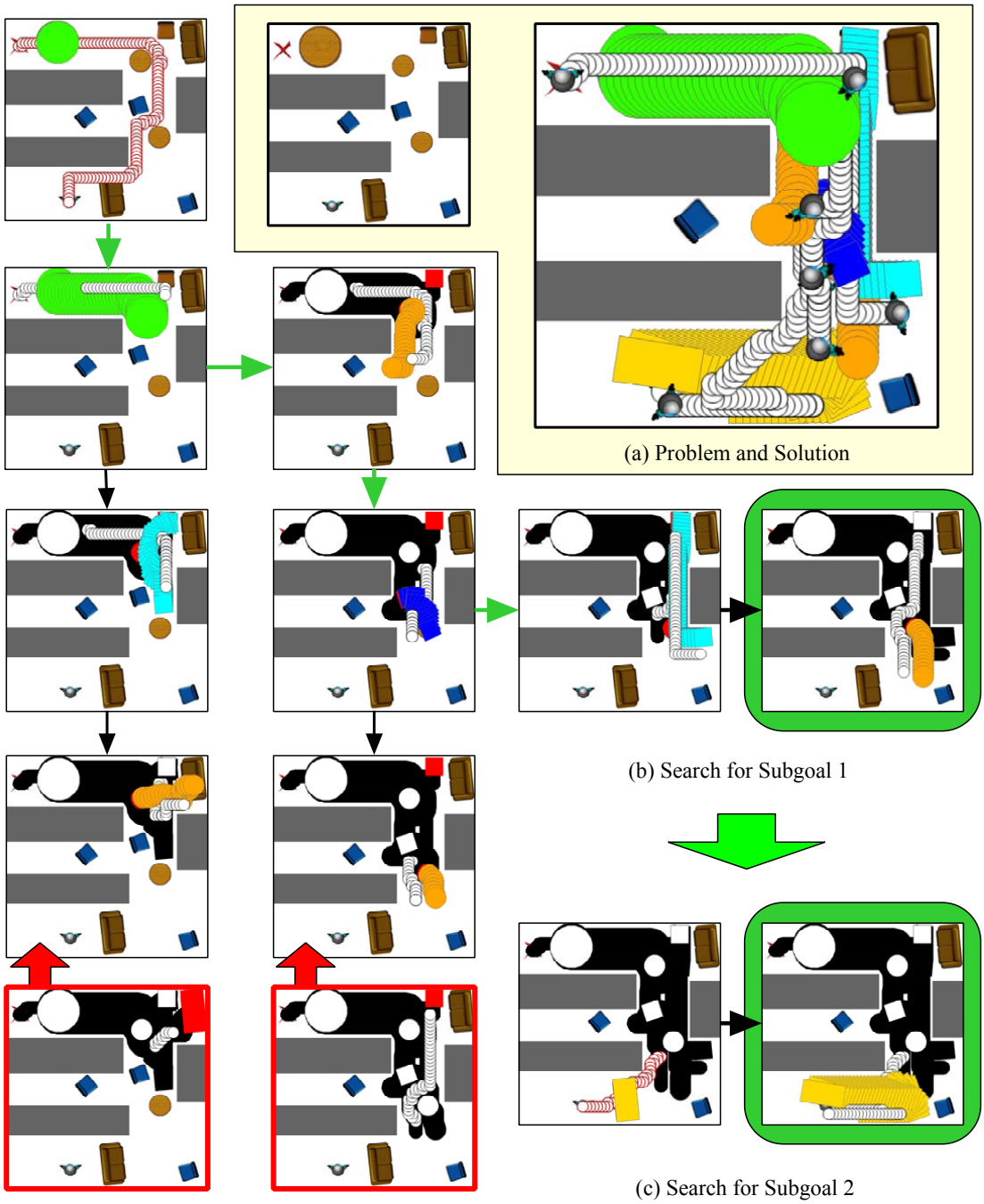


Figure 6.6: A search tree for the given example. Large upward arrows indicate backtracking when an object cannot be resolved.

pretability. □ indicates a successful base case while (NIL) reflects backtracking.

6.6 Implementation

The algorithm described in this chapter is general for two and three dimensional spaces with arbitrary configuration spaces for the manipulator. In this section we interpret the algorithm for our two dimensional NAMO domain giving details on planning, experimental results and complexity analysis.

6.6.1 Planning Details

When constructing a NAMO plan, we directly apply the algorithm in Section 6.5 by selecting a computational representation of paths and artificial constraints:

- For paths, we choose a grid planner based on an evenly spaced discretization of $\mathcal{C}_R$. The robot configuration space has three dimensions: $(\mathbb{R} \times \mathbb{R} \times SO(2))$. Robot paths are planned in a matrix of resolution $(10cm, 10cm, 10^o)$ in each dimension respectively. This yields a simple, resolution complete search space.

- In the two dimensional domain, artificial constraints are sets of points in $\mathbb{R}^2$. Due to the rotation of objects, these sets could have complex curved boundaries. To reduce constraint verification (collision detection) to polygon intersection, we define swept volumes using a local convex hull approximation method similar to [82, 83]. We construct local bounding polygons for the object and robot throughout the path.

- All obstacles and artificial constraints are rasterized in the form of an occupancy grid of the environment. Set membership in world coordinates is confirmed by verifying the occupancy of grid cells.

Choosing a *Navigate* path (τ_1) in $\mathcal{C}_R$ is performed using A^*. The heuristic is euclidian distance with a penalty for entering $\mathcal{X}_R(O_i)$ for the first instance of O_i along the path. This heuristic is selected to minimize the number of objects that will violate the artificial constraint in the preceding plan.

	Figure 6.1	Figure 6.3	Figure 6.5	Figure 6.6
# Objects	4	4	4	9
# Manipulated	4	2	2	6
Planning Time	0.77s	0.05s	0.10s	2.08s

Table 6.1: Quantities of objects and running times for examples in figures.

Analogously, since *Manipulate* paths (τ_2) have no explicit goal, we use best first search to make a selection. The first path and resulting state encountered by the search that satisfy the artificial constraints are chosen by the planner. Heuristically, we penalize states where robot or the Manipulated object enter movable configuration space obstacles.

6.6.2 Results

The implemented planner was tested on a number of examples, including all the figures presented in this paper. Table 6.6.2 summarizes the running times on an Intel Pentium M 1.6Ghz processor.

Of the presented examples, Figures 6.1, 6.5 and 6.6 cannot be solved by previous planners [25, 31, 84]. In Figure 6.3, the proposed method is asymptotically faster than SELECTCONNECT due to the early selection of *Navigate* paths as constraints in contrast to path validation during *Manipulate* search. However, this choice precludes completeness in the proposed implementation. For L_1 problems, SELECTCONNECT will discover remote *Navigate* paths that are not considered by the proposed implementation.

We find these results encouraging towards the implementation of this planner on a real robot system. Since the planner searches locally in the configuration space of the robot, the same algorithm can be applied directly to very high dimensional configuration spaces by replacing grid search methods with sampling-based alternatives.

6.6.3 Complexity

Since IDENTIFY-OBSTACLE never considers an obstacle more than once at any level of the search tree, it can generate at most $m!$ sequences. Each sequence can contain m objects

to be resolved by RESOLVE-CONSTRAINTS. A breadth first search of $\mathcal{C}_R$ of resolution d in n dimensions has runtime $O(d^n)$. The overall algorithm is asymptotically $O(m!d^n)$. This is a vast overestimate. In most cases only a few sequences will satisfy the conditions of the planner.

Notice, however, that each of three "Choose" statements in RESOLVE-CONSTRAINTS is an opportunity for backtracking (Lines 4, 6 and 8). Selecting a different simple path for *Manipulate* or *Navigate* will yield distinct artificial constraints for the remainder of the search. While enumerating all possible simple paths for robot motion and manipulation is computationally expensive, selecting a subset of these paths may prove to be valuable.

6.7 Summary

So far in this thesis we have developed a configuration space representation for domains with movable obstacles. We used this representation to formulate two algorithms. First, we considered the interactions between *Navigate* and *Manipulate* to create SELECTCONNECT, a resolution complete planner for an intuitive class of problems. Next, we introduced reverse search and artificial constraints to handle the interactions between *Manipulate* spaces of distinct objects. Our results show that both algorithms can be implemented for real-time applications in complex, realistic environments with many obstacles.

Future work will consider the possibility of reducing the number of object orderings and examining alternative object paths. Some likely classes of heuristics are the following:

Accessibility Constraints - Currently we search through all orderings of objects that violate an artificial constraint. However, clearly some objects cannot be reached by the robot before others are moved. These objects must be moved at a later time-step.

Path Heuristics - Reverse search carries significant advantages to forward search in selecting alternative paths. Simply by finding paths that explore distinct, or distant, portions of space we would change the topology of artificial constraints and therefore open distinct possibilities for prior object placements.

In addition to the investigation of heuristics, it will be interesting to study the potential for using artificial constraints to determine the necessity of intermediate states. Doing so will enable planners to address the greater challenges of non-monotone problems.

7

3D Movable Obstacles

7.1 Broader Applications

The NAMO algorithms we described in Chapter 3 and 6 were presented in a general configuration space framework. We chose this representation to make the algorithms useful across a wide range of robots and tasks. For illustration we focused primarily on the application of a mobile robot and an environment with large movable obstacles. In this chapter we go beyond planar examples and implement NAMO planning for a redundant articulated manipulator that can displace 3D obstacles. We will show that the principles of NAMO planning remain intact and address unique challenges in higher dimensions. Further information on this algorithm can be found in [85] and [86].

To motivate our discussion of 3D manipulation, consider a mobile manipulator that acts as a service robot in a mechanic's workshop. The robot is asked to retrieve a particular tool or machine part. Just like in *Navigation* tasks, the robot cannot obtain the part without first opening cabinet doors, moving additional objects and making space for the desired manipulation. For instance, Figure 7.1 shows a simulated robot reaching for a hammer by first displacing the box fan and the gear. Similar examples can be found in a variety of service settings including home assistance, medical care and other indoor tasks that require fine manipulation.

From the perspective of higher level goal selection, the structure for solving the pro-

Figure 7.1: Mobile manipulator moves the gear and fan before retrieving the hammer.

posed manipulation problem follows directly from our discussion of NAMO planning. Primarily, we use artificial constraints described in 6. However, this application also exhibits a number of new challenges: a more complex robot configuration space, constrained object placement and constrained object motion. The first two challenges will be described and resolved in this chapter. The third will be described in Chapter 8.

7.2 Domain Observations

Before introducing the differences between the three dimensional and our previous experiments let us first identify the similarities. As before, we have a robot manipulator with n degrees of freedom, a set of rigid body static obstacles $\mathbf{O}_F = \{F_1, \ldots, F_f\}$ and a set of rigid movable objects $\mathbf{O}_M = \{O_1, \ldots, O_m\}$. The robot can change its own configuration or *Navigate* and *Manipulate* to change both its configuration and that of an object. Hence, collision-free motion, Eq. 3.2.4 - 3.2.9, transfer directly to this application.

The first difference between our previous problem specification and the one posed in this chapter is that the desired motion is manipulation for a target object rather than robot navigation. Furthermore, instead of manipulating planar polygons, the robot now interacts with polyhedra that are assigned a set of properties:

Geometry - One closed triangular mesh.

CENTER OF MASS - A point in the local object frame.

GRASP SET - End-effector workspace transforms in the object frame.

CONSTRAINTS - Specification of permitted motion in workspace coordinates.

Each object has a workspace configuration q consisting of a translation and orientation. The robot configuration r is defined in joint space. We are given the initial configuration of the robot and all movable obstacles: $W^0 = (0, r^0, q_1^0, q_2^0, \ldots, q_m^0)$ and a final configuration q_G^{goal} for movable obstacle O_G. The robot must construct a sequence of joint paths that result in the desired object placement.

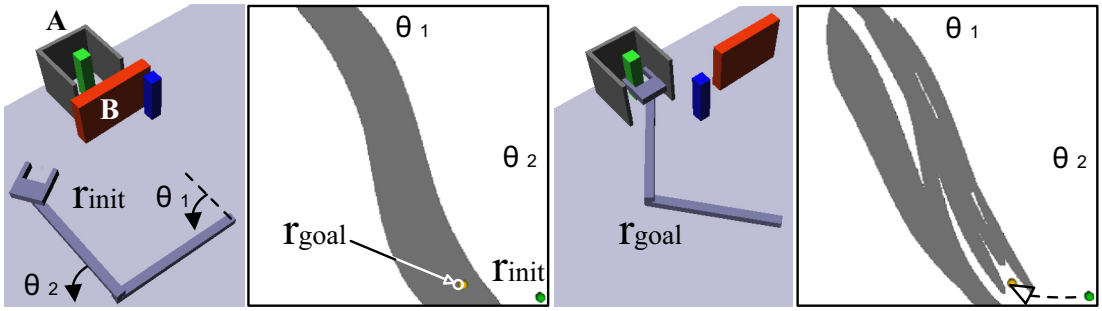
7.2.1 Simplifying Assumptions

For the purposes of computational efficiency and algorithmic clarity we present methods that are applicable to a subset of domain problems. First, we restrict the action space to rigid grasp manipulation. Hence, a *Manipulate* path for an object always keeps a fixed transform between the end-effector and the object. Second, we assume that the problem is *monotone* or that if a solution exists, it can be found by moving each obstacle once. Consequently we need not consider plans longer than the number of movable obstacles.

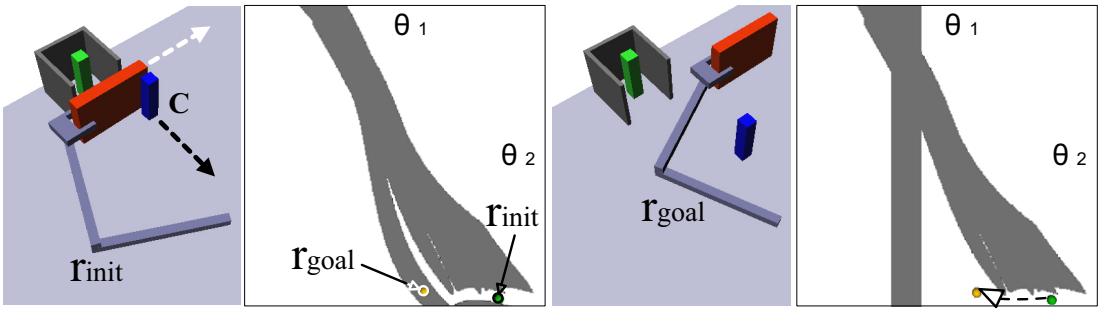
We further constrain object placements to variations of initial configurations that alter position and rotation about the z-axis. The objects must be placed on a flat surface to establish planar contact under the COG. While movable objects may be stacked initially, they can only be placed on fixed objects by the robot. These are sufficient but not necessary conditions for static stability. They allow us to pre-sample the space for possible placements without regard for object shape or the displacement of potential surfaces.

7.2.2 Challenges

Despite our assumptions the computational complexity of our domain remains exponential as described in Chapters 6 and 2. In fact, this domain magnifies the difficulty of representing a configuration space that changes with the displacement of objects. Furthermore, articulated manipulation introduces additional challenges for selecting object



(a) Configuration space of the first two robot links before and after box B is moved.



(b) Configuration space of the robot grasping B before and after C is moved.

Figure 7.2: Simulated workspace and configuration space

placements and manipulation paths.

First we look at the configuration space of an articulated manipulator and show that the heuristics developed in previous chapters are especially relevant. The example in Figure 7.2 shows the interaction between *Navigate* and *Manipulate* operators. The robot is required to reach box O_A in the cubicle. Grey areas on the right hand side represent configuration space obstacles or configurations that are in collision. Notice that the goal is inside a configuration space obstacle. In Figure 7.2(a), displacing object O_B creates the desired free $\mathcal{C}_R$ space with a nonlinear deformation.

Similarly, there is significant interaction between distinct *Manipulate* operators. Figure 7.2(b) illustrates the configuration space of the robot grasping O_B. For interpretability we only allow translations of O_B and summarize $\mathcal{C}_R$ with the first two joints. Again the target displacement is inside an obstacle. Moving O_C creates space which allows the target displacement. Unlike planar domains, even translations of cubes create complex

deformations in the robot's free space. These changes prevent us from reducing the problem to a free space graph, however they do not inhibit search methods such as constraint relaxation and artificial constraints introduced in Chapter 6.

While our NAMO heuristics help us overcome the $O(m!(pe)^m)$ complexity, estimated in Chapter 6, we now attend to path validation and object placements. Previously, we assumed that path validation can be performed in constant time, e, however searching the configuration space of an articulated manipulator is particularly expensive, especially in the case of redundant joints [39]. Recent planning research focuses solely on the problem of single object manipulation by an articulated robot [17, 87, 18, 43]. Additionally, in the 3D obstacle domain placements, p, do not correspond to arbitrary object configurations. We first need to determine locations that constitute stable object displacements. Subsequently, all valid displacement paths must terminate in one of these stable configurations.

7.3 Algorithm

In domains with articulated robots we observed that commonly object goals lie in configuration space obstacles. As a result, rather than explicitly planning to establish *spatial connectivity* as described in previous chapters, we focus on using future motion to restrict the space of valid displacements for blocking objects. This approach increases the interpretability of our algorithm for 3D manipulation and allows us to focus on the specific challenges of this domain.

The last step of the plan is always to $Manipulate\ O_G$ to its goal configuration along some path τ. If no objects are moved from their initial configurations, $Manipulate(\tau, O_G)$ would collide with a set of objects: $\mathbf{O}_{PAST}$. Our planner identifies these objects and plans to displace them. Since these displacements may also be blocked, the set $\mathbf{O}_{PAST}$ is expanded to include indirectly blocking objects. While there may be infinite possible paths for object displacement, there is a finite number of object orderings. For a particular choice of paths, the number of objects that must be moved is relatively small. Hence, we search the space of orderings while incrementally selecting object placements and paths

```
RESOLVESPATIALCONSTRAINTS(O_c, O_PAST, O_FUT, r^{t+2}, V)
 1  (r^t, GRASP) ← PLANGRASP(O_c)
 2  P ← FINDPLACEMENTS(O_c, GRASP, O_FUT, V)

 3  M(τ_M, O_c) ← PLANMANIP(O_c, r^t, P, O_FUT)
 4  O_FUT ← O_FUT. append O_c

 5  N(τ_N) ← PLANNAVIGATION(τ_M(1), r^{t+2}, O_FUT)
 6  V ← V append SWEPTVOLUME(τ_M, τ_N)

 7  O_PAST ← O_PAST append IDBLOCK(τ_M, τ_N)
 8  if O_PAST = ∅
 9     then return ◇

10  for  each O_P in O_PAST
11  do if RESOLVESPATIALCONSTRAINTS
12        (O_P, O_PAST − O_P, O_FUT, r^t, V)
13        then return ◇
14  return NIL
```

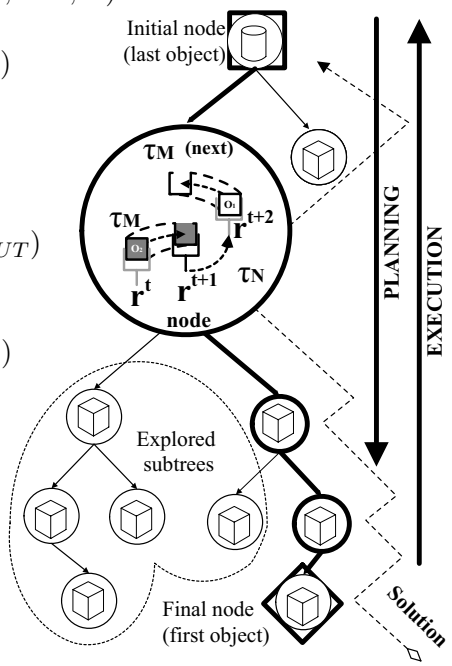


Figure 7.3: RSC Algorithm pseudo-code and diagram.

that are valid with regard to future *Navigation* and *Manipulation*.

To formalize the search we let $\mathbb{V}$ be the volume of space that must be unoccupied to validate future operators. After sampling $Manipulate(\tau, O_G)$, $\mathbb{V}$ contains the workspace volume that is occupied by the robot and O_G during the continuous execution of τ. All objects that collide with $\mathbb{V}$ are placed in $\mathbf{O}_{PAST}$ and must be displaced to configurations that do not collide with $\mathbb{V}$. Recursively, paths that *Manipulate* these objects and *Navigate* to the subsequent subgoals define volumes of workspace that are added to $\mathbb{V}$ to constrain the placement of prior obstacles.

Our algorithm RESOLVESPATIALCONSTRAINTS(RSC) is detailed in Figure 7.3. RSC follows the a depth first search over orderings of the blocking obstacles. The algorithm returns NIL and backtracks if any of the three PLAN operations fail (Lines 1,3 and 5). Each node in the diagram represents a sampled *Manipulation of* some object and *Navigation from* grasping the object in its final configuration to grasping the subsequent object in its initial configuration. The children of the node are choices for the directly preceding object

displacement. The algorithm is initialized by specifying the goal object, O_G, and a goal configuration for the robot, r^{t+2}. The first call to RSC specifies these two parameters and empty sets for O_{PAST}, O_{FUT} and $\mathbb{V}$. The planner terminates a branch when PLANGRASP, PLANMANIPULATION or PLANNAVIGATION are unsuccessful. It also backtracks when all the orderings represented by a node's children terminate unsuccessfully.

7.4 Motion Sampling

In order to apply the RSC algorithm we are required to choose methods for sampling object placements and robot paths for *Navigate* and *Manipulate* operators. Our strategy for selecting placements samples a uniform distribution of stable configurations in the environment. With regard to paths, our previous strategy of using A* search becomes impractical when considering the joint space of an articulated manipulator. Currently, the fastest strategies for manipulation planning use randomized methods for selecting configurations. Ahuactzin introduces Ariadne's Clew [87], Simeon applies Probabilistic Road Maps [18] while LaValle and Kuffner develop Rapidly-Exploring Random Trees (RRT)[39]. Since we are interested in changing and changeable environments, we focus on single query planning. Namely, we apply a variant of RRT due to its recent experimental success in many applications and simple implementation. The remainder of this chapter will detail our algorithms for sampling placements and paths.

7.4.1 Sampling Placements

First, let us consider choosing potential placements for objects that interfere with our path to the goal. Our criterion for static stability simply requires that the object be placed with its center of mass (COM) above any environment surface. The lower surface of the object must make contact with the environment surface surrounding the projected COM. Since all objects are modeled as triangular meshes, the set of placements consists of random points on triangles, $\mathcal{T}$, with upward facing normals, $\mathbf{n}(\mathcal{T}_j)$. It is important to select placements uniformly from the entire surface area since the areas of $\mathcal{T}_j$ may differ.

(a) Collision free placements (b) Placements with a valid grasp

Figure 7.4: Autonomously selected placements for the drill.

The area of the sample space, S, is a sum of bounding box areas for the triangles $\mathbf{b}(\mathcal{T})$. For a conditional impulse δ, we have:

$$|S| = \sum_{O_i \in \mathbf{O}_F} \sum_{\mathcal{T}_j \in O_i} \mathbf{area}(\mathbf{b}(\mathcal{T}_j))\delta_{(\mathbf{n}(\mathcal{T}_j)=[0\ 0\ 1]^T)} \tag{7.4.1}$$

We rejection sample a placement by randomly choosing a triangle according to its boxed contribution to S and selecting a point from a uniform distribution $U(S)$ over $\mathbf{b}(\mathcal{T})$. The probability distribution of a point p is:

$$P(p) = U(S) = \frac{\mathbf{area}(\mathbf{b}(\mathcal{T}(p)))U(\mathcal{T}(p))}{|S|} \tag{7.4.2}$$

Each sampled point decides the translation of the placement and a separate random draw is used for orientation around the z-axis. Since objects may only be placed on static surfaces, we initialize the algorithm by sampling a set of potential object placements.

The call to FINDPLACEMENTS(O_c, GRASP, $\mathbf{O}_{FUT}$, $\mathbb{V}$) positions O_c at each of the sampled placements subject to three constraints. First, the object may not be in collision with a static object or a swept volume of a future *Navigation* or *Manipulation* path.

Second, there must be a valid inverse kinematics solution, r^{t+1}, for the GRASP previously selected in PLANGRASP. Third, the robot in configuration r^{t+1} may not collide with fixed obstacles or elements of $\mathbf{O}_{FUT}$ which are fixed in their initial configurations due to the monotone assumption. The function returns a set of grasping configurations that satisfy these criteria.

7.4.2 Sampling Paths

Any node in the RSC tree shown in Figure 7.3 corresponds to the displacement of some object O_c. The displacement is defined by three reference robot configurations: r^t, r^{t+1} and r^{t+2}. r^{t+2} is the grasping configuration for the parent object O_p. r^t and r^{t+1} are grasping configurations for O_c in its initial and final configurations respectively. Since O_p is scheduled to be displaced immediately after O_c, r^{t+2} is known. PLANGRASP, PLANMANIP and PLANNAVIGATION are three calls to a motion planning algorithm that are used to select r^t,r^{t+1} and sample paths $\tau_M(r^t, r^{t+1})$ and $\tau_N(r^{t+1}, r^{t+2})$ that connect them.

The motion planner we use is an extension of RRT. The basic RRT algorithm shown in Figure 7.4.2 generates paths by growing a tree from a start configuration. At each iteration, the planner selects a random configuration that lies within robot joint limits and detects its nearest neighbor in the tree. Subject to collision detection, the planner attempts takes a step from the nearest neighbor towards the random configuration. Direct application of the RRT algorithm to our domain would not be successful since the expected time of convergence to an arbitrary point is infinity. Instead, we use the RRT-Connect heuristic, which attempts to merge a start and goal tree by using the leaves of one as samples for the other.[39]

Notice that there is one challenge in applying a point-to-point planner to our domain. Due to the monotone assumption, r^t and r^t+2 are known to be grasping configurations of the current and subsequent object in their initial states. However, r^t is not known. This is the grasping configuration for O_c after it has been displaced. While we might simply choose some configuration, there is no guarantee or even heuristic to indicate that there

```
RRT(r_init)
1  T.init(r_init)
2  for k = 1 to K
3  do r_rand ← RandomConfig;
4     r_near ← NearestNeighbor(r, T);
5     if Extend(r_rand, r_near, r_new)
6        then AddEdge(T, r_near, r_new)
7  return NIL
```

```
Extend(r, r_near, r_new)
1  r_dir ← Unit(r, r_near);
2  if Norm(r, r_near) < ε
3     then r_new ← r;
4     else  r_new ← r_new + ε × r_dir;
5  return CheckCollision(r_new)
```

Figure 7.5: Basic RRT Algorithm Pseudo Code

exists a valid *Manipulate* path that results in the desired displacement. In Chapter 6 we solved this problem by choosing the first valid object configuration found by the planner. We now show how the same concept can be applied in randomized search.

Our algorithm uses Multi-Goal RRT-Connect (MG-RRT) as shown in Figure 7.4.2. This approach is similar to one that was previously presented by Hirano[88]. The search consists of three subroutines. Instead of choosing a particular goal configuration from which to grow a goal tree, we create a set of goal trees starting at all sampled goals. For each goal tree, the algorithm proceeds as follows:

The start tree is extended as a typical RRT. If a new leaf is generated, the leaf is used as a sample for the goal tree. The Connect routine iterates calls to Extend until either the goal tree merges with the start or the expansion encounters an obstacle. Subsequently, the start and goal trees are swapped, allowing the goal tree to grow randomly and attempting to Connect the start tree. During each iteration of MG-RRT, the algorithm repeats this procedure for each of the goal trees.

Our planner yields a natural selection process for goal configurations. The goal tree which connects to the start fastest is selected to be the final object configuration and the

```
MultiGoal-RRT-Connect(r_init, r_G1, ..., r_GN)
1  T^S.init(r_init)
2  for i = 1 to N
3  do T^Gi.init(r_Gi);
4  for k = 1 to K
5  do for i = 1 to N
6      do if ConnectTwoTrees(T^S, T^Gi) = ◇
7            or ConnectTwoTrees(T^Gi, T^S) = ◇
8            then return Path(T^S, T^Gi)
9  return NIL
```

```
ConnectTwoTrees(T^A, T^B)
1  r_rand ← RandomConfig;
2  r_near ← NearestNeighbor(T^A, r_rand);
3  if Extend(r_rand, r_near, r_new)
4    then AddEdge(T^A, r_near, r_new)
5         if Connect(T^B, r_new) = ◇
6           then return ◇;
7  return NIL
```

```
Connect(T, r)
1  r_near ← NearestNeighbor(r, T);
2  while Extend(r, r_near, r_new) = true
3  do r_near = r_new;
4     if r_new = r
5       then return ◇;
6  return NIL;
```

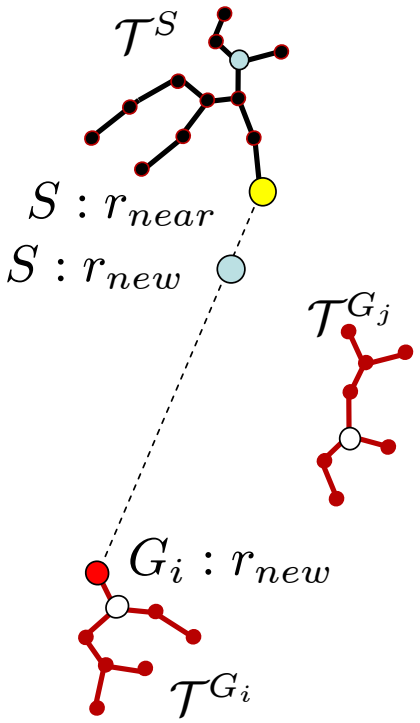


Figure 7.6: Multi-Goal RRT-Connect pseudo code.

path is chosen as the sample. Let us consider how this planner is applied in the context of RSC function calls. For all plans the planner is not allowed to collide with static obstacles or obstacles in $\mathbf{O}_{FUT}$, whose initial configuration remains static until after O_c is displaced.

PlanGrasp - decides r^t, the grasp of O_c: This is a MultiGoal plan from r^0, the initial configuration, to any robot configuration that grasps O_c. This step ensures that the grasp can be reached by the robot from the initial state given static objects.

PLANMANIPULATION - decides r^{t+1}: This MultiGoal plan is given the starting configuration r^t and the associated GRASP of O_c. Its goal is to connect r^t to any valid configuration in the set preselected by FINDPLACEMENTS (Section 7.4.1). The returned plan is the path τ_M.

PLANNAVIGATION - The final path planner is a single goal RRT-Connect that identifies τ_N. The planner searches from the final grasp of O_c: $\tau_M(1) = r^{t+1}$ to the grasp of the next object r^{t+2}.

Since RRT trees grow indefinitely when a solution is not found, we limit the depth of expansion. RSC backtracks when any of the three plans are unsuccessful.

Observe that in each call to PLAN, we accept the first successful connection as the path sample. Therefore, the RRT tends to connect objects to nearby goal configurations. This choice is reasonable but not exhaustive. When all obstacle orderings have been attempted, one should consider choosing different path samples. We propose two methods that avoid using the same paths and placements in subsequent trials: removing goals from MG-RRT and biasing RRT search away from previous solutions.

7.5 Constraints

In addition to the standard collision detection that is required for motion planning, RSC search must handle other forms of constraints. One simple example of higher-order constraints is an ordering constraint on the motion of supporting objects. Obstacles that are initially supported by other movable objects must be moved prior to manipulating their support. Two of the more complex constraints are the placement constraints that result from reverse search and the motion constraints on real world objects.

7.5.1 Placement Constraints

The RSC algorithm is based on sampling future motions and using the resulting paths as constraints for previous motion. Namely, $\mathbb{V}$ is a swept volume of the space occupied

by the future motions of the robot and displaced obstacles. This volume must be cleared of all objects for the sampled path to be valid. $\mathbb{V}$ serves two purposes: to detect objects that collide with sampled paths and to constrain the displacement of these objects. When searching for object placements, we ensure that the objects do not collide with the swept volumes.

In our current implementation, $\mathbb{V}$ is represented simply as a discrete sequence of robot and object models. This is reflected in the running time of placement search (Section 7.6). When generating these constraints, future work should consider local convex hull approximations or occupancy grid methods to reduce the collision checking involved in placement selection. An important topic is to ensure the safety of the volumes. While overestimating the size of a volume may lead to the removal of some valid placements, underestimating it would mean that the planner constructs invalid paths.

7.5.2 Motion Constraints

The second type of constraint handled by our planner restricts motion of scene objects. Each object is associated with a transformation that serves as a reference frame for the task constraint. A boolean vector determines which degrees of freedom are can be changed for a valid displacement. For instance, cabinet doors only permit rotation about the z-axis.

To maintain the random sampling of paths, we considered modifications of the RRT algorithm for constrained objects. Our approach to constraints sampling is detailed in Chapter 8. When applied in the context of RSC search, our final planner efficiently determined solutions to problems with constrained objects and had no effect on planning for unconstrained motion. One example can be seen in Figure 7.7 where the robot must open a cabinet in order to reach for the drill.

Figure 7.7: Simulated 8-DOF manipulator delivers an obstructed drill.

7.6 Results

In order to test our approach, we performed simulated experiments with a three-link planar robot arm and an 8-DOF mobile manipulator based on the 6-DOF PUMA arm (Fig. 7.1,7.7,7.8). The planar environment shown in Figure 7.8 consists of five rectangular movable objects and one fixed cubicle. The PUMA was placed in a simulated workshop environment with various shapes of tools and mechanical objects. This scene has 28 objects, 16 of which are movable. In the workshop some objects such as doors were constrained to only move along or about fixed axes.

For each scenario we performed 100 experiments. Table 7.1 summarizes the average total planning time in seconds, t_{total}, the average number of objects moved, $\overline{m}$, and individual times for each operation of RSC. $(t_{sv}, t_{place}, t_M, t_N, t_{grasp})$ correspond to the time spent creating collision models for swept volumes, identifying valid placements, sampling valid *Manipulate*, *Navigate* and *Grasp* paths.

Table 7.2 shows the average proportion of time spent on each subtask over all three experiments. *RRT* and *Place* represent the cumulative cost for sampling paths and

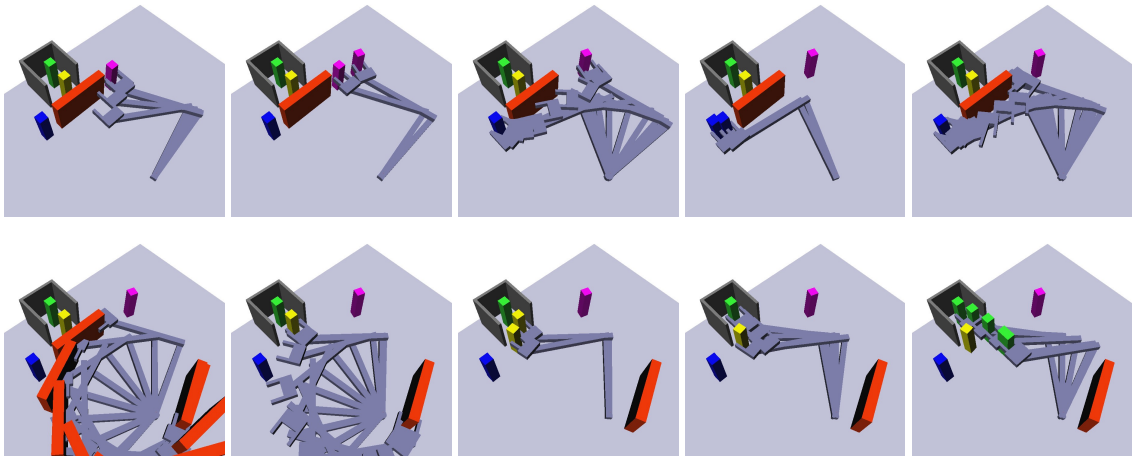


Figure 7.8: Simulated planar arm retrieves a green block from the cubicle.

Exp.	$\overline{m}$	t_{total}	t_{rrt}	t_{sv}	t_{place}	t_M	t_N	t_{grasp}
Figure 7.1	3.0	13.12	6.55	1.12	1.75	7.52	1.06	1.15
Figure 7.7	3.8	13.26	4.56	3.16	0.55	5.92	1.30	1.69
Figure 7.8	4.6	7.15	2.63	1.68	0.97	1.10	2.38	0.71

Table 7.1: RSC: Run times for algorithm components

Total	*RRT*	*SV*	*Place*	*M*	*N*	Grasp
100%	46.7%	22.9%	11.8%	44.9%	20.6%	12.5%

Table 7.2: Average cost for each procedure over all experiments.

placements throughout planning. In both tables subtasks are grouped when their time measurements are independent.

Overall, we found that RSC consistently finds solutions to monotone problems in seconds of planning time. Due to the sampling strategy, the algorithm does not require significant overhead for high dimensional problems. One clear distinction is that for Figure 7.8 sampling τ_N takes longer than τ_M. This contrasts Fig. 7.1, 7.7. Faster manipulation search is reasonable since τ_M has multiple feasible goals. In the detailed scenarios PlanManipulation is more expensive due to collision and motion constraints on complex manipulated objects.

7.7 Summary

The primary goal of this chapter was to show how our methods for NAMO planning could be generalized beyond planar domains. We achieved this with the particularly important and challenging example of 3D manipulation by articulated robots. Despite the increased complexity for path planning and the additional constraints we showed that a planner could solve complex problems in seconds.

In order to achieve efficient search, we introduced the RSC reverse search algorithm to handle exponentially complex non-linear configuration spaces. These spaces are common in manipulation due to the interactions between the robot, the movable objects and the environment. The lower level implementation of RSC required us to develop strategies for sampling object placements and paths. We gave a simple placement sampling approach and presented MG-RRT, an algorithm for searching paths that connect starting object configurations to a set of valid potential placements.

This chapter has also hinted at another important aspect of planning interaction with arbitrary obstacles. Constraints on obstacles place constraints on paths that the robot can use to displace them. In the next chapter we will address the problem of manipulation planning for obstacles such as doors and drawers that cannot be moved arbitrarily.

8

Constrained Objects

8.1 Introduction

The final topic of this thesis is manipulation planning for constrained objects. This topic became particularly relevant during the development of Chapter 7. First, when adding realistic objects to the scene it became apparent that the robot should not plan to unhinge doors or tip volatile containers. Second, our method for efficient joint space search made it challenging to enforce such constraints during planning. If a planner selects from a known set of actions, we can simply restrict the actions to those that satisfy the object constraints. However for articulated robots we have seen that randomized planners are better suited for efficient computation. This poses a significant problem since for many constrained tasks the probability of selecting a random action that lies on the constraint manifold is small or zero. In this chapter we explore the application of randomized motion planning algorithms to problems where the robot path is required to obey workspace constraints.

Task compliance or constrained motion is critical in situations where the robot comes in contact with constrained objects. Consider Figure 8.1 and Figure 8.10. Notice the global deviations from straight paths as the robot first pushes the door away, goes around the fan, and finally brings the door open to full swing. Many real-world tasks ranging from opening doors and pushing carts to helping align beams for construction and removing rubble exhibit workspace constraints. In these circumstances, the robot must not

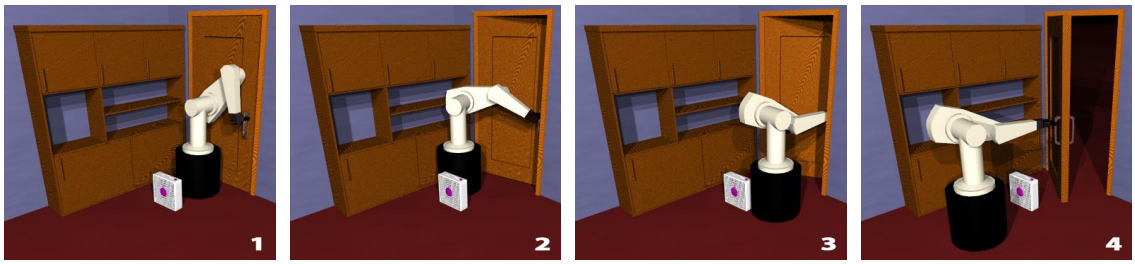


Figure 8.1: DOOR: Computed solution to door opening avoids obstacles and joint limits.

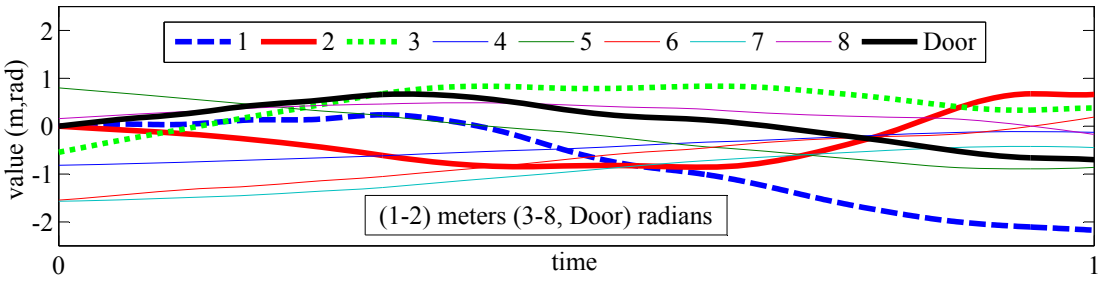


Figure 8.2: DOOR: Paths for robot joints and door rotation about the z-axis.

only preserve the task constraint, but also avoid collisions and joint limits throughout a planned motion. Redundant robots, such as mobile manipulators and humanoids, have the dexterity to accomplish both objectives. The challenge is to efficiently explore alternative joint space paths that satisfy task constraints without being trapped in local minima.

To address constrained joint space planning, we first present background for our study and identify the key characteristics of the problem and the proposed solution. Section 8.4 develops a simple and general representation of constraints based on [89] and [90]. This representation is shown to be well suited for a wide range of problems in motion planning. Sections 8.5 and 8.6 describe three joint space sampling techniques that preserve any constraint expressed with our representation. Section 8.7 experimentally evaluates the three techniques in simulation. The comparison includes different tasks, choices for time step and error tolerance. Finally, we discuss extensions to our proposed tools and directions for future research.

8.2 Related Work

In addition to finding a collision-free joint space path [26], our problem requires that each configuration along the path satisfy workspace constraints for the end-effector [89],[90]. We distinguish this problem from specifying a single goal pose for the end-effector [91] or a path that the end-effector must follow [92, 93, 94]. In our case, the end-effector path is not predetermined and the constraints must be satisfied throughout the path.

Even when the end-effector path is specified, handling robot redundancy poses a difficult challenge. Typically, redundancy resolution is performed with local [95, 96] or global [97] techniques for optimal control. These methods optimize configuration-dependent criteria such as manipulability. Obstacle distance, a common criterion for collision avoidance, has a highly non-linear relationship to joint configurations and leads to the use of local optimization [98, 99, 100]. These methods require the generation of distance/potential functions for obstacles. Moreover, they do not find solutions in the presence of local minima that exist in all three of our examples.

For a more comprehensive exploration of the search space, motion planning research has headed towards feasible solutions and probabilistically complete algorithms such as PRM[38] and RRT [101][39]. These algorithms operate by generating random joint space displacements. However, in the case of constrained motion, the probability of randomly choosing configurations that satisfy the constraints is not significant or zero. This is shown for problems involving closed chains in [102] and [103].

To address this challenge, some approaches use domain-specific attributes such as closed chain structure [102, 103, 104] or dynamic filtering [105]. Among these techniques, one algorithm, Randomized Gradient Descent (RGD)[102] has been extended for use with arbitrary constraints [106]. RGD randomly shifts sampled configurations in directions that minimize constraint error. However, [106] applies this algorithm with general constraints on a case-by-case basis. We propose to adapt the existing task frame formalism [89] to unify the representation of constraints and initiate the comparison of algorithms for constrained sampling.

In addition to RGD, we considered ATACE [106] as well as adapting the path-following strategies from [93, 94]. However, ATACE explores task space motions with RRTs and locally follows them in joint space. This approach cannot solve Figure 8.1, since more than one joint space path must be explored along a single task space path. [93] and [94] require a partition of the robot into "base" and "redundant" joints. Error due to perturbations of redundant joints is compensated by inverse kinematics of the base joints. This algorithm relies on significant freedom for base joints, since obstacle and joint limit constraints will prevent error compensation even when redundancy in the remaining joints is available.

This chapter presents two contributions in the domain of constrained joint space planning. First of all, we apply an intuitive formulation of task space constraints. Secondly, we propose and evaluate a strategy for generating joint space samples that satisfy these constraints. Our strategy is shown to be experimentally successful in comparison to RGD with regard to different tasks and variations in parameter choices.

8.3 Preliminaries

This chapter uses three spaces of coordinates:

$\mathbf{r}_i$ Joint space coordinates refer to a vector of single-axis translations and rotations of the robot joints.

$\mathbf{T}_B^A$ Work space homogeneous matrices represent the rigid transformation of frame $\mathcal{F}^B$ with respect to frame $\mathcal{F}^A$.

$\mathbf{x}_i$ Task space coordinates will be used for computing error with respect to the task frame (Section 8.4).

Note that task space is equivalent to work space by rigid transformation. The distinction allows for a simple formulation of constraints. We also employ $\mathbf{R}_B^A$ for rotations of $\mathcal{F}^B$ with respect to $\mathcal{F}^A$. Likewise, $\mathbf{p}^A$ and $\mathbf{z}^A$ are vectors in frame A.

8.4 Specifying Constraints

We define a task constraint as a restriction on the freedom of motion of a robot end-effector.[89][90] The degrees of freedom for a rigid body are defined by translations and rotations in a given coordinate frame. The task frame, $\mathcal{F}^t$, is derived from the world frame, $\mathcal{F}^0$, by a rigid transformation of the world axes. The matrix $\mathbf{T}_t^0$ specifies the position and orientation of $\mathcal{F}^t$ with respect to the world frame.

In the task frame we have various options for quantifying end-effector error. For instance, translations may be in Cartesian or Spherical coordinates, while rotations could be described by Euler Angles or Quaternions. The choice of coordinates will affect the types of constraints that we can represent. [90] For any representation we define $\mathbb{C}$, the *motion constraint vector*. $\mathbb{C}$ is a vector of binary values for each of the coordinates. A value of one indicates that the end-effector motion may not change the coordinate. We can also represent $\mathbb{C}$ as a diagonal selection matrix [89]:

$$\mathbb{C} = \begin{bmatrix} c_1 \\ \dots \\ c_n \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} c_1 & & \\ & \dots & \\ & & c_n \end{bmatrix} \tag{8.4.1}$$

We use $\mathbb{C}$ and $\mathbf{C}$ interchangeably. Without loss of generality we highlight the example of a Cartesian representation of translations and roll/pitch/yaw rotations about fixed axes. In these coordinates, a rotation with respect to the task frame is defined about the x_t, y_t and z_t axes of $\mathcal{F}^t$:

$$\mathbf{R}_B^t = R(z_t, \phi)R(y_t, \theta)R(x_t, \psi) \tag{8.4.2}$$

This choice of coordinates directly specifies permitted translations and rotations about the task frame axes. The $\mathbb{C}$ vector for our representation is six dimensional:

$$\mathbb{C}_{RPY} = [\ c_x \quad c_y \quad c_z \quad c_\psi \quad c_\theta \quad c_\phi\ ]^T \tag{8.4.3}$$

complete representation of a task constraint consists of the task frame $\mathcal{F}_t$, the coordinate system and the motion constraint vector $\mathbb{C}_t$. The description may be constant throughout the motion plan, or it could vary in accordance with the robot configuration or the state of a higher-level planner. Let us consider the spectrum of constraint definitions that have intuitive formulations in our representation. The first two constraint types are not only the simplest but also directly applicable to the NAMO domain presented in Chapter 7. A single constraint vector for each object is sufficient to plan compliant interaction.

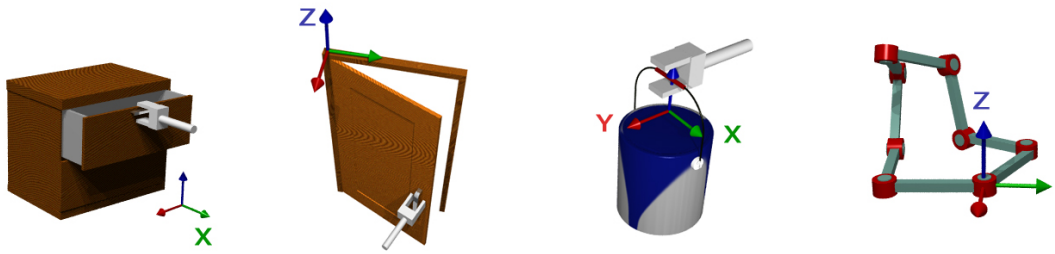
Fixed Frames: The simplest task defines a single frame $\mathcal{F}_t$ and $\mathbb{C}_t$ for the entire path plan. Fixed frame constraints occur when manipulating objects that are kinematically linked to the environment (Fig. 8.1, 8.8). $\mathcal{F}^t$ is any frame in which the axes align with the directions of constrained motion. Usually the transformation describing $\mathcal{F}^t$, $\mathbf{T}_t^0$ is the position and orientation of the object. $\mathbb{C}_t$ indicates which axes of $\mathcal{F}_t$ permit valid displacements. When planning constrained manipulation, one can assume that the object is rigidly attached to the robot during grasp. At the time of grasp, let $\mathbf{T}_A^0$ and $\mathbf{T}_e^0$ represent the position and orientation of object A and the end-effector respectively. We define the grasp transform $\mathbf{T_G}$:

$$\mathbf{T_G} = \mathbf{T}_A^e = \mathbf{T}_0^e \mathbf{T}_A^0 = (\mathbf{T}_e^0)^{-1} \mathbf{T}_A^0 \tag{8.4.4}$$

During manipulation, the kinematics of the end-effector are extended to include the manipulated object such that $\mathbb{C}$ directly specifies the permitted displacements of the object.

$$\mathbf{T}_{e'}^0(\mathbf{q}) = \mathbf{T}_e^0(\mathbf{q}) \mathbf{T_G} \tag{8.4.5}$$

Simple Frame Parameters: Fixed frames are a global constraint on the robot end-effector. Suppose the constraint is defined locally with respect to the position of the end-effector or another function of the planner state. For instance, a task may consist of



(a) $[\,0\,1\,1\,1\,1\,1\,]^T$ (b) $[\,1\,1\,1\,1\,1\,0\,]^T$ (c) $[\,0\,0\,0\,1\,1\,0\,]^T$ (d) $[\,1\,1\,1\,1\,1\,0\,]^T$

Figure 8.3: Roll/pitch/yaw constraints. (c-d) are parameterized by robot configuration.

manipulating a sequence of constrained objects. Each object is assigned a distinct task frame and constraint vector. The planner selects the task frame based on the object with which it is in contact.

Alternatively, the constraint for a single object may be defined locally with respect to the configuration of the robot. For example, a local constraint on rotation is meaningful for transporting open containers of liquid such as paint (Figure 8.9). In this case, the task frame orientation is designated by the world frame and the position is determined by the end-effector configuration. The constraint vector remains a specification of the directions of permissible motion.

Kinematic Closure: An important constraint for multi-arm manipulation and reconfigurable robots is a linkage with a closed kinematic chain. One approach to specifying kinematic closure is to break the chain at an arbitrary joint $\mathbf{j}_k$. The linkage becomes an open chain with a constraint defined by the freedoms of the detached joint $\mathbf{j}_k$.

We formulate closure as a special case of parameterized task frame constraints. For any open chain joint configuration, $\mathbf{q}$, the kinematic expression for $\mathbf{j}_k$ along each chain yields a work space transform $\mathbf{T}(\mathbf{q})$ describing $\mathbf{j}_k$ with respect to $\mathcal{F}^0$. One of these transforms can be specified as the task frame, $\mathbf{T}_t^0(\mathbf{q})$ while the others are end-effectors $\mathbf{T}_e^0(\mathbf{q})$. As with previous examples, the constraint vector intuitively specifies the degrees of freedom for the joint $\mathbf{j}_k$.

Constraint Paths and Surfaces: Some elaborate constraints may require the end-effector to remain on a non-linear path or maintain contact with a complex surface. In this case there are at least three options:

1. Parameterize the task frame by the nearest position on the path or surface to the sampled configuration.
2. Parameterize the surface by a subset of standard orthogonal coordinates and constrain the remaining coordinates to the surface values.
3. Define a coordinate system that implicitly encodes distance from the surface as a change to the coordinates.

Notice that in the case where a path is parameterized by time, the end-effector trajectory is a continuous sequence of task frame parameters. Extending the search space with time, the frame of a sampled configuration is decided by time.

8.5 Introducing Constrained Sampling

In selecting a representation of task constraints we identified a linear subspace of constrained configurations for the end-effector in the task space. This is the space of all coordinates x in $\mathcal{F}^t$ such that $x_i = 0$ when $c_i = 1$. This space maps to a non-linear manifold in robot joint space. When sampling the joint space, randomized planners typically produce samples that lie outside the constraint manifold. Our proposed methods use $\mathbb{C}$ to formulate a distance metric in task space and project samples within a tolerance distance (ϵ) of the constraint.

8.5.1 Computing Distance

Having identified a sampled configuration $\mathbf{r}_s$, we compute the forward kinematics for $\mathbf{r}_s$. Commonly the end-effector frame, $\mathcal{F}^e$, is found in homogeneous coordinates as the

transformation $\mathbf{T}_e^0(\mathbf{r}_s)$. The displacement of $\mathcal{F}^e$ with respect to the task frame $\mathcal{F}^t$ is:

$$\mathbf{T}_e^t(\mathbf{r}_s) = \mathbf{T}_0^t \mathbf{T}_e^0(\mathbf{r}_s) = (\mathbf{T}_t^0)^{-1} \mathbf{T}_e^0(\mathbf{r}_s) \tag{8.5.1}$$

We now represent the end-effector displacement with respect to the task frame in task coordinates. The Appendix also defines this change in representation for roll/pitch/yaw.

$$\Delta\mathbf{x} \equiv \mathbf{T}_e^t(\mathbf{r}_s) \tag{8.5.2}$$

The task space error is found simply by the product in Eq. 8.5.3. This product has the effect of selection presented in Eq. 8.5.4.

$$\Delta\mathbf{x}_{err} = \begin{bmatrix} e_1 & \dots & e_n \end{bmatrix} = \mathbf{C}\Delta\mathbf{x} \tag{8.5.3}$$

$$e_i = \begin{cases} 0 & c_i = 0 \\ \Delta\mathbf{x}_i & c_i = 1 \end{cases} \tag{8.5.4}$$

8.5.2 Baseline: Randomized Gradient Descent

The proposed distance metric detects when a configuration satisfies the constraint tolerance. Furthermore, it can be used to identify task space motions that reduce error. Our algorithms use this information to construct constrained samples.

The three methods we compare are Randomized Gradient Descent (RGD) [102][106], Tangent-Space Sapmling (TS), and First-Order Retraction (FR). For simplicity, we will describe each approach as a modification to the basic RRT algorithm summarized in Figure 8.4. The algorithm samples a random configuration $\mathbf{r}_{rand}$ and finds its nearest neighbor in the tree $\mathbf{r}_{near}$. The sampled configuration $\mathbf{r}_s$ is placed at a fixed distance Δt along the vector from $\mathbf{r}_{near}$ to $\mathbf{r}_{rand}$.

As a baseline, consider the RGD algorithm detailed in Figure 8.6. After computing the task space error of $\mathbf{r}_s$, this method continues to uniformly sample the neighborhood

```
TASK_CONSTRAINED_RRT(r_init, Δt)
 1  T.init(r_init);
 2  for a = 1 to A
 3  do r_rand ← RANDOM_CONFIG;
 4     r_near ← NEAREST_NEIGHBOR(r_rand, T);
 5     r_dir ← (r_rand − r_near)/|r_rand − r_near|;
 6     r_s = r_near + r_dir Δt;
 7     if Constrained_NEW_CONFIG(r_s, r_near)
 8        then T. add_vertex (r_s);
 9             T. add_edge (r_near, r_s);
10  return T
```

```
COMPUTE_TASK_ERROR(r_s, r_near)
1  (C, T_0^t) ← RETRIEVE_CONSTRAINT(r_s, r_near);
2  T_e^0 ← FORWARD_KINEMATICS(r_s);
3  T_e^t ← T_0^t T_e^0;
4  Δx ← TASK_COORDINATES(T_e^t);
5  Δx_err ← CΔx
6  return Δx_err;
```

Figure 8.4: Pseudo-code for Task-Constrained RRT (TC-RRT) construction.

of the configuration within a radius d_{max}. The error of each new configuration $\mathbf{r}'_s$ is compared with $\mathbf{r}_s$. If the error is less, $\mathbf{r}'_s$ is substituted for $\mathbf{r}_s$. The procedure terminates when the error is less than ϵ or the maximum iteration count is met.

8.6 Relating Joint Motion to Constraint Error

Due to random selection, the RGD algorithm will typically evaluate forward kinematics for a large number of configurations that result in greater task space error. To avoid this computation, we identify the relationship between task space error and joint space motion. Since the relationship is nonlinear, we use a first-order approximation.

The basic Jacobian, $\mathbf{J}^0$, is a matrix of partial derivatives relating joint space velocities to end-effector linear and angular velocities. We compute the task frame Jacobian, $\mathbf{J}^t$, and use its inverse to find joint space displacements that resolve task space error. $\mathbf{J}^0$, can be found analytically given the kinematics of each joint at a sampled configuration $\mathbf{r}_s$ of

the robot. [107] Let $\mathcal{J}_{Pi}$ and $\mathcal{J}_{Oi}$ be translational and rotational contributions of joint i:

$$\mathbf{J}^0 = \begin{bmatrix} \mathcal{J}_{P1} & \cdots & \mathcal{J}_{Pn} \\ \mathcal{J}_{O1} & \cdots & \mathcal{J}_{On} \end{bmatrix} \tag{8.6.1}$$

Typically, this computation is performed in the world frame $\mathcal{F}^0$. We invert the orientation of the task frame, $\mathbf{R}_t^0$, and transform the Jacobian into $\mathcal{F}^t$ as follows:

$$\mathbf{J}^t = \begin{bmatrix} \mathbf{R}_0^t & 0 \\ 0 & \mathbf{R}_0^t \end{bmatrix} \mathbf{J}^0 \tag{8.6.2}$$

The lower three rows of $\mathbf{J}^0$ and $\mathbf{J}^t$ map to angular velocities. However, angular velocities are not instantaneous changes to any actual parameters that represent orientation. Given the configuration $\mathbf{r}_s$, instantaneous velocities have a linear relationship $\mathbf{E}(\mathbf{r}_s)$ to the time derivatives of many position and orientation parameters as given in the Appendix.[90]

$$\mathbf{J}(\mathbf{r}_s) = \mathbf{E}(\mathbf{r}_s)\mathbf{J}^t(\mathbf{r}_s) \tag{8.6.3}$$

Having established a mapping of instantaneous motion between joint and task space we invert this relationship. For redundant manipulators, there are many joint motions that map to a single task space displacement. Our algorithms use the right pseudo-inverse, $\mathbf{J}^\dagger$, which maps instantaneous position error in frame $\mathcal{F}^t$ to the *least-norm velocities* in joint space required to correct it.

$$\mathbf{J}^\dagger = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1} \tag{8.6.4}$$

While there are many algorithms for computing the pseudo-inverse, we have used LU decomposition, a fast $O(n^3)$ operation. This method is faster than iterative SVD and more commonly available than the Greville method [108].

The mapping between joint space and task space allows the introduction of two simple

```
RGD_New_Config(r_s, r_near)
 1  i ← 0;   j ← 0;
 2  Δx_err ← Compute_Task_Error(r_s, r_near);
 3  while i < I and j < J and |Δx_err| > ε
 4  do i ← i + 1;   j ← j + 1;
 5      r'_s = r_s + Random_Displacement(d_max);
 6      Δx'_err ← Compute_Task_Error(r_s, r_near);
 7      if Δx'_err < Δx_err
 8        then j ← 0;   r_s = r'_s;   Δx_err = Δx'_err;
 9      if Δx_err ≤ ε
10        then if In_Collision(r_s)
11                then return false;
12                else return true;
13  return false;
```

Figure 8.5: Pseudo-code for Randomized Gradient Descent sampling.

```
TS_New_Config(r_s, r_near)
1  Compute_Task_Error(r_s, r_near);
2  (C, T_0^t) ← Retrieve_Constraint(r_s, r_near);
3  J ← Jacobian(r_near);
4  r_near);
5  r_err ← J†Δx_err;
6  Δq = r_s − r_near;
7  Δq' = Δq − J†CJΔq;
8  r_s ← r_near + Δq';
9  return RGD_New_Config(r_s, r_near);
```

Figure 8.6: Pseudo-code for Tangent Space Sampling

techniques for constrained sampling. Tangent Space Sampling (TS) projects each RRT sample into the linear tangent space of its nearest neighbor. First-Order Retraction (FR) iteratively approximates the minimal displacement that removes task space error and applies it to the sample.

8.6.1 Tangent Space Sampling

First, observe that any existing $\mathbf{r}_{near}$ in the RRT is within tolerance of the constraint manifold. For closed chains, [109] finds that small joint displacements from $\mathbf{r}_{near}$ that are

tangent to the constraint manifold have a higher probability of also being within tolerance. In our case, these displacements have no instantaneous component in the direction of task error. In other words, for displacement $\Delta\mathbf{q}$ and Jacobian $\mathbf{J}(\mathbf{r}_{near})$,

$$\mathbf{CJ}\Delta\mathbf{q} = \mathbf{0}. \tag{8.6.5}$$

To use this insight in RRT search, let $\mathbf{r}_s$ be a sampled configuration at a small displacement $\Delta\mathbf{q}$ from $\mathbf{r}_{near}$. We project the displacement into the null space of the task constraint.

$$\Delta\mathbf{q}' = (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{CJ})\Delta\mathbf{q} \tag{8.6.6}$$

Eq. 8.6.6 represents the least-norm change to $\Delta\mathbf{q}$ that places it in the tangent space. Notice that $\Delta\mathbf{q}'$ is distinct from the minimal joint motion that achieves an equivalent task space displacement $\mathbf{J}\Delta\mathbf{q}'$. The random direction is largely preserved. The projected sample is simply $\mathbf{r}'_s = \mathbf{r}_{near} + \Delta\mathbf{q}'$. Due to the non-linearity of the constraint manifold, $\mathbf{r}'_s$ is still unlikely to be within error tolerance. RGD is applied to the sample to further reduce task space error.

8.6.2 First-Order Retraction

Although similar in form to TS, First-Order Retraction behaves more like RGD in that it iteratively displaces the sample towards the constraint manifold. Unlike RGD, the displacement is not random but directed towards removing constraint error. Unlike TS, the Jacobian is computed at the sampled configuration $\mathbf{r}_s$ rather than $\mathbf{r}_{near}$.

Consider the retraction of a single configuration $\mathbf{r}_s$. First, we find the task space error, $\Delta\mathbf{x}_{err}$, according to Section 8.5.1. If the error exceeds tolerance, we compute $\Delta\mathbf{r}_{err}$, the least-norm joint displacement that compensates and adjust the sample to $\mathbf{r}'_s$:

$$\Delta\mathbf{r}_{err} = \mathbf{J}^{\dagger}\Delta\mathbf{x}_{err} \tag{8.6.7}$$

$$\mathbf{r}'_s = \mathbf{r}_s - \Delta\mathbf{r}_{err} \tag{8.6.8}$$

```
FR_NEW_CONFIG(r_s, r_near)
 1  r_r ← r_s
 2  Δx_err ← COMPUTE_TASK_ERROR(r_s, r_near);
 3  while |Δx_err| > ε
 4  do RETRACT_CONFIG(r_s, Δx_err);
 5       if |r_s − r_r| > |r_r − r_near|
 6          then return false;
 7       Δx_err ← COMPUTE_TASK_ERROR(r_s, r_near);
 8  if IN_COLLISION(r_s)
 9     then return false;
10  return true;

RETRACT_CONFIG(r_s, Δx_err)
1  J ← JACOBIAN(r_s);
2  Δr_err ← J†Δx_err;
3  r_s ← (r_s − Δr_err); r_err|);
```

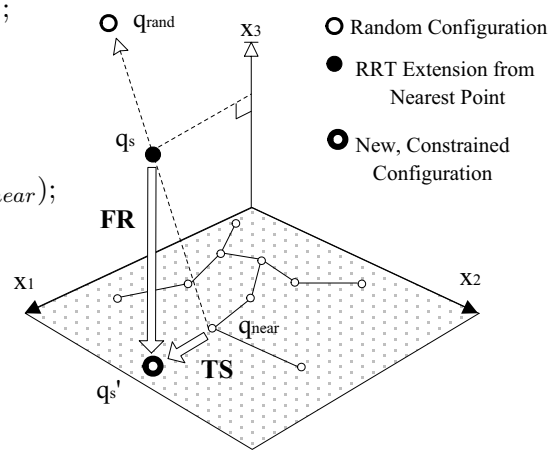


Figure 8.7: FR_NEW_CONFIG Algorithm pseudo-code and diagram.

Since the Jacobian is a first-order approximation, task space changes do not linearly map to changes in the joint space. Locally, we apply gradient descent as shown in Figure 8.6.1. Close to singularities the pseudo-inverse may become unstable. To resolve this, we discard samples when the magnitude of adjustment exceeds the original displacement.

8.7 Results

We evaluated the sampling strategies on three sets of simulated experiments: DOOR, DRAWER and PAINT. The task is to plan a path for a PUMA 560 manipulator on a holonomic mobile base. The base adds two redundant degrees of freedom (dof) to the six-dof PUMA. In each example, the robot is given an initial grasping configuration from which we grow an RRT according to each constraint method. The robot must maintain the task constraint and achieve a goal that satisfies a workspace criterion:

Door - Open the door past 0.6 rad.

Drawer - Extend the drawer 0.3 m.

Paint - Move the paint 6.0 m to the right.

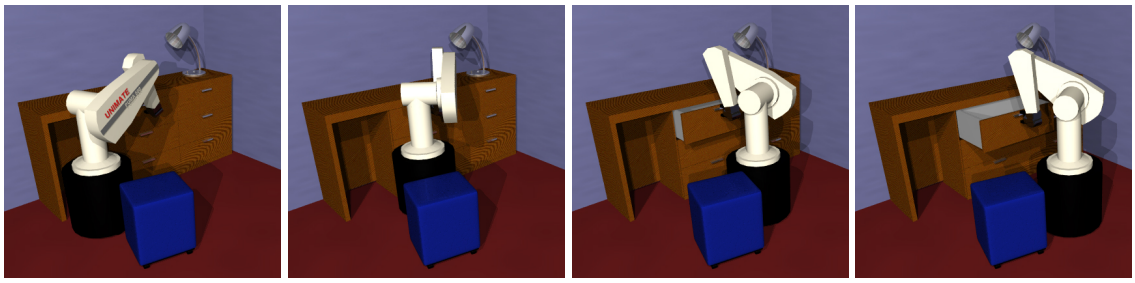


Figure 8.8: DRAWER: Solved drawer opening with obstacle avoidance.

Once a configuration is found that satisfies the constraint, the search is considered successful. During the search, the door only rotates at the hinge, the drawer only slides and the paint is not allowed to pitch or roll. Figures 8.1-8.9 show successful solutions where the robot employs redundancy to avoid obstacles and joint limits. For instance, in 8.9, joint 4 raises the arm to avoid debris, then lowers under the ladder. Joint 7 moves accordingly, maintaining the constraint.

To show the relative stability of our methods, we conducted experiments with different choices for step size, Δt, and error tolerance, ϵ. Each trial was performed 10 times and terminated when it was unsuccessful after 10 minutes. Both FR and TS required no additional parameter choices. For RGD, we set $I = 1000$, $J = 100$, $d_{max} = \Delta t/10.0$. These parameters resulted in the highest overall performance on the three examples in our preliminary testing.

For efficiency, the RRTs in all experiments used the VCollide collision checking package and the kd-tree nearest neighbor approximation.[110] For simplicity, the basic RRT algorithm was used without goal biasing of the samples. Computation was performed on an Intel T2600 2.16GHz processor, with an average memory load of 40MB. Final trajectories were smoothed with cubic splines. Tables 8.1-8.3 summarize the average computation time when all 10 trials succeeded. When at least one trial failed, the tables show the percentage of successful trials. The tables are blank when all 10 trials did not result in a solution within the allocated 10 minutes per trial.

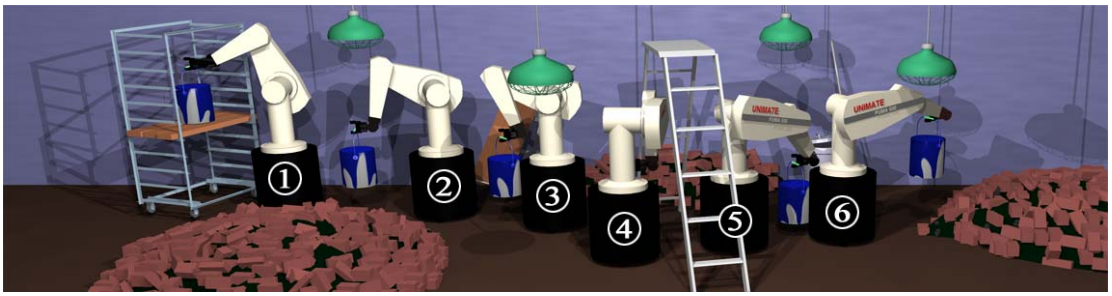


Figure 8.9: PAINT: Simulation requiring upright transport of paint.

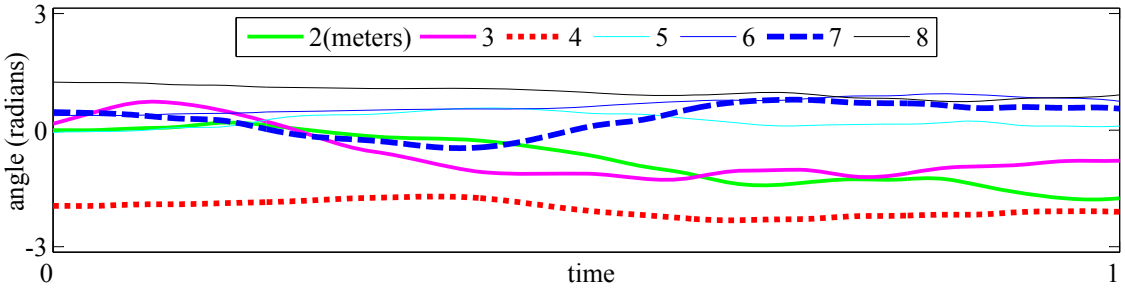


Figure 8.10: PAINT (FR): Paths for joints 2-8 in Paint experiment.

Experiments DOOR and DRAWER favor FR in terms of both computation time and stability. In these examples, we see that RGD and TS perform optimally with the largest error tolerance and a step size of approximately .02. Lower performance when $\Delta t = .04$ is most likely caused by the distance of the sample from the constraint manifold. For RGD, this distance implies a larger number of iterations to reach tolerance. In the case of TS, a linear approximation to the constraint manifold is less accurate for larger displacements.

For large tolerances, TS and RGD outperform FR in Experiment PAINT. This is likely due to two factors: the significantly larger space of valid configurations and its linearity. Since only two coordinates are constrained, random samples have a higher probability of being within tolerance. Furthermore, translations of the base joints map directly to unconstrained coordinates. Observe that even in a lightly constrained setting, only FR succeeds for lower tolerances.

Overall, we found that Jacobian based algorithms required less computation and performed comparably with RGD in the worst conditions. FR showed significantly more

Table 8.1: Runtimes for Experiment 1: Door

		Δt (Step Size)				
	$\epsilon(m, r)$	.04	.02	.01	.005	.0025
RGD	10^{-4}	**107.16**	**50.16**	**69.56**	**130.57**	**256.0**
	10^{-5}					60%
	10^{-6}					
TS	10^{-4}	**22.60**	**11.35**	**18.33**	**25.03**	**62.22**
	10^{-5}				90%	**114.77**
	10^{-6}					
FR	10^{-4}	**4.46**	**10.28**	**14.64**	**36.92**	**139.25**
	10^{-5}	**12.70**	**20.28**	**15.21**	**44.74**	**89.49**
	10^{-6}	**5.56**	**8.82**	**19.50**	**63.86**	**108.8**

Table 8.2: Runtimes for Experiment 2: Drawer

		Δt (Step Size)				
	$\epsilon(m, r)$	.04	.02	.01	.005	.0025
RGD	10^{-4}	**21.42**	**13.98**	**19.49**	**40.09**	**96.91**
	10^{-5}				**282.98**	90%
	10^{-6}					
TS	10^{-4}	**7.81**	**5.38**	**9.92**	**21.24**	**43.37**
	10^{-5}				**153.65**	**68.62**
	10^{-6}					
FR	10^{-4}	**1.58**	**4.53**	**6.05**	**19.52**	**34.44**
	10^{-5}	**2.50**	**3.40**	**6.18**	**16.84**	**33.72**
	10^{-6}	**1.64**	**4.65**	**8.18**	**17.21**	**32.45**

invariance with respect to error tolerance. We also observed an approximately linear relationship between computation time and time step. This increase is unavoidable since smaller time-steps increase the length of the motion plan.

8.8 Extensions

Having introduced and compared a set of algorithms for joint space planning with workspace constraints, we now consider some simple extensions to our framework.

Unilateral Constraints: The motion constraint vector $\mathbb{C}$ represents bilateral constraints, prohibiting positive and negative coordinate change. Suppose a constraint re-

Table 8.3: Runtimes for Experiment 3: Paint

		Δt (Step Size)					
	$\epsilon(m, r)$	.08	.04	.02	.01	.005	.0025
RGD	10^{-4}	**18.88**	80%	**119.19**	80%	50%	10%
	10^{-5}	50%	80%	70%	50%	60%	10%
	10^{-6}						
TS	10^{-4}	**10.78**	**28.67**	**57.69**	**131.61**	80%	10%
	10^{-5}	60%	**293.88**	**186.94**	**209.14**	90%	1%
	10^{-6}						
FR	10^{-4}	**20.21**	**45.08**	**127.24**	**288.47**	60%	20%
	10^{-5}	**20.42**	**60.51**	**137.53**	90%	70%	10%
	10^{-6}	**22.36**	**42.95**	**126.94**	80%	30%	

quires a coordinate to remain within an interval or to change monotonically. These cases can be treated as either obstacles or parameterized constraints.

In the case of the former, samples that violate the constraint will be discarded. The latter option specifies $c_i = 1$ or $c_i = 0$ for coordinate i based on whether or not the sampled configuration violates the boundary. This choice will generate more valid samples, yet it may bias these samples towards the constraint bounds.

Alternative Planning Strategies: The algorithms evaluated in Section 8.7 were all based on the single-query RRT algorithm. However, both RGD and FR are well suited for use in PRMs among other multi-query algorithms. As long as the constraint distance metric is well-defined throughout the space, arbitrary configurations can be displaced to the constraint manifold. All three algorithms can be used to connect PRM samples.

The generality of our methods extends to the use of heuristics such as RRT-Connect during search [39]. Since the only modification to RRT is in the NewConfig method, most heuristic variants of RRT are trivially extended to include the proposed strategies.

Multiple Constraints: Some tasks require multiple end-effector constraints. This can be achieved with a composite constraint vector $\mathbb{C}_M$. Let $\mathbb{C}_i$ represent the constraint

vector for task i. Then we have:

$$\mathbb{C}_M = [\ \mathbb{C}_1^T \ \Big|\ \mathbb{C}_2^T \ \Big|\ \cdots \ \Big|\ \mathbb{C}_n^T\ ]^T \tag{8.8.1}$$

For the TS and FR algorithms, we also construct a generalized Jacobian matrix composed by stacking the individual task Jacobians. The computation of task space distance and the planning algorithms remain unchanged.

Abstract End-Effectors: The generalization of task definitions also extends to abstract end effectors. For fixed frames we displaced the end-effector frame to the grasped object. In fact, any coordinate defined as a function of one or more robot links can be constrained. An important example is the robot center of mass (COM). The COM position is a linear combination of the positions $\mathbf{p}_{mi}$ of the individual link masses, m_i:,

$$\mathbf{p}_{COM} = \sum_{i=0}^{n} m_i \mathbf{p}_i / \sum_{i=0}^{n} m_i \tag{8.8.2}$$

This definition is sufficient to compute task space error. For TS and FR we also define the COM Jacobian as follows:

$$\mathbf{J}^0_{com} = \begin{bmatrix} \mathcal{J}_{P1} & \dots & \mathcal{J}_{Pn} \end{bmatrix} \tag{8.8.3}$$

$$\mathcal{J}_{Pi} = \begin{cases} (\frac{1}{M}\sum_{j=i}^{n} m_j)\mathbf{z}_{i-1} & \text{(prismatic)} \\ \frac{1}{M}\sum_{j=i}^{n} m_j(\mathbf{z}_{i-1} \times (\mathbf{p}_{mj} - \mathbf{p}_{i-1})) & \text{(revolute)} \end{cases}$$

When the end-effector is a non-linear function of the joints, the Jacobian can be approximated numerically by computing kinematics for small displacements in each joint.

8.9 Summary

In this chapter we aimed to enable randomized joint space planning for manipulating constrained objects with minimal representational requirements for object constraints. To

accomplish this, we presented a unified representation for task space constraints in the context of joint space motion planning. We described three algorithms for task constrained sampling in joint space. Our comparison of the algorithms indicated that First-Order Retraction is typically faster and significantly more invariant to step size and error tolerance than alternative techniques. Finally, we generalized our approach to variations in constraint definitions and algorithm choices.

When implementing the NAMO planner in Chapter 7, we chose FR-RRT for its stability and efficiency. To enable object constraints we simply associated a constraint vector with each object and applied the planner extension when handling a constrained object. Our modified algorithm had no effect on the performance of search during interaction with unconstrained objects. For constrained objects the results showed similar runtimes to the ones presented in this chapter.

Our experimental findings regarding the efficiency of Jacobian based algorithms also contribute to plan execution. Computing the Jacobian during planning allows us to measure the manipulability of sampled configurations: $det(\mathbf{J}\mathbf{J}^T)^{1/2}$ [111]. Maintaining a manipulability threshold will permit stable use of local compliance or impedance control and allow for error while following a computed path. In addition to hard constraints discussed in Section 8.8, future work might also consider task projection into the null space of $\mathbf{J}^\dagger$ could be used to prioritize constraints.[112, 113]

Of course, soft constraints are one of many possible deeper explorations of constrained search. Our proposed tools demonstrate the feasibility of constraining efficient joint space search and simplify the process of including constraints during global NAMO planning.

9

Conclusion

In this thesis we showed that moving obstacles out of the way is not only a useful goal for robots but also a practical challenge that can be addressed with existing hardware and computation. Our demonstration included a full development cycle from motion planning to implementation on a humanoid robot. We presented an efficient strategy for selecting objects and identifying helpful displacements. We investigated controls for balance and manipulation. Finally, we combined these efforts in a complete system that recognized environment objects and executed Navigation Among Movable Obstacles.

Our continue work in motion planning allowed us to consider more complex object interaction, three dimensional domains and the problem of interacting with constrained objects. In each case we presented new algorithms and evaluated their performance in simulation. We believe that continued work along these lines will identify even more powerful solutions to NAMO problems and enable greater autonomy for service robots.

Currently, this thesis work only scratches the surface of potential interactions between robots and their environments. Many exciting problems remain for future research. These include further investigation of NAMO strategies such as the selection of alternative paths during manipulation. With regard to execution, we are interested in more robust methods for handling uncertain situations. Finally, we seek to extend this work to new forms of interaction as well as automated detection and use of structure in new environments.

A

Dynamic $\mathcal{C}$-space Modification

The two algorithms described in our work use the MANIP-SEARCH(W^t, O_1, C_2) routine as defined in Section 3.5.2. MANIP-SEARCH is required to search the available robot action space for manipulating O_1. The search should terminate when $\mathcal{C}_R^{free}$ contains a path in W^{t+2} from r^{t+2} to any $r_2 \in C_2$. Conceptually this is a clear objective that is easily extended to higher dimensional spaces. In terms of implementation, deciding whether the goal has been satisfied is an interesting challenge.

Suppose the robot displaces obstacle O_1 along some path. We need to quickly determine whether a path (r^{t+2}, C_2) exists. In terms of implementation and computational difficulty it seems daunting to formulate/maintain a complex model of all C_i. We avoid this by depicting the entire $\mathcal{C}_R$ as a planar grid, where configuration space obstacles are represented by rasterizing their perimeter.[114] Since the robot embodies a cell in the $\mathcal{C}$-space, we can trivially test for connectivity between r^{t+2} and C_2 by locally searching for a path between the grid cell corresponding to r^t and any $r_2 \in C_2$.

We are left with the problem of updating the $\mathcal{C}_R$ in a way that would facilitate goal testing. Updating the discrete grid involves rasterizing the perimeter of the obstacle in its original configuration to *remove* it, and rasterizing it again in the new configuration. The difficulty lies in the fact that $\mathcal{C}$-space obstacles can overlap. *Removing* an obstacle requires us to clear the cells that belong to the obstacle. Due to overlap, however, a single

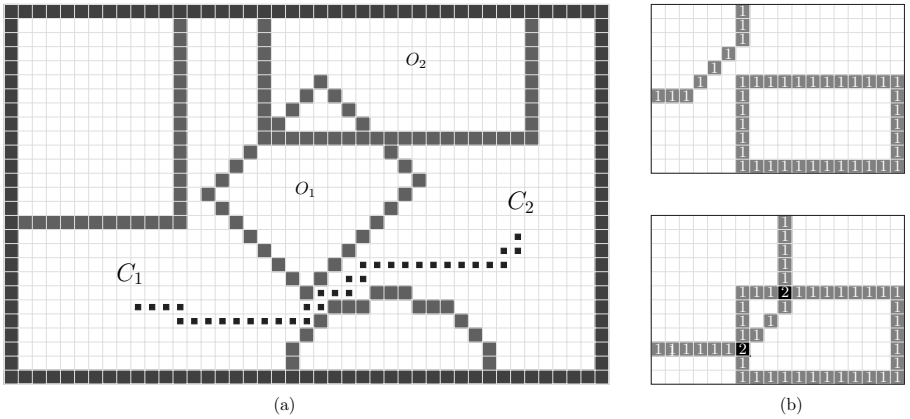


Figure A.1: Update of dynamic $\mathcal{C}_R$ by reference counting.

grid cell may belong to more than one obstacle and should not be cleared when only one obstacle is removed. Simple binary grids provide no tools for determining if there are other objects occupying a given cell. Naive extensions such as maintenance of obstacle lists for each grid cell can add another dimension to $\mathcal{C}_R$. Alternatively, testing obstacles for inclusion involves unnecessary computation.

We solve this problem as follows: rather than maintaining a binary grid to represent $\mathcal{C}_R$, we keep an integer grid as shown in Figure A.1(b). Each cell is represented by a counter of the number of obstacles to which it belongs. *Removing* an obstacle decrements the cell counters during rasterization, and *adding* the obstacle increments them. The robot can then reach all accessible cells of 0 count. This algorithm requires no added complexity in implementation and allows us to perform $\mathcal{C}$-space updates in time linear to the perimeter of the manipulated obstacle.

Our current implementation relies on local search that either finds a path as shown in Figure A.1(a) or fails after exploring all accessible cells. It is also possible to encode more information into grid cells to indicate the bordering component of free space. During rasterization, a verification procedure to detect free space opening may replace the search.

B

Preview Control

Chapter 4 identified a linear relationship between the state of a humanoid robot's COM, x_r, and its center of pressure or *zmp*. We ensure dynamic stability by computing an x_r trajectory which ensures that the *zmp* trajectory stays at the center of the robot's feet. The x_r trajectory should be maximally smooth and should result in close tracking of *zmp*. These are competing objectives since desired *zmp* changes abruptly due to changes in supporting feet. Earlier methods have used mixtures of pre-designed trajectories and more costly gradient descent techniques [115, 116, 117, 118, 119]. Recently, Kajita proposed preview control (PC) as an alternative solution [46]. This appendix summarizes the theory of preview control and applies it to biped locomotion.

Preview control, described by Tomizuka and Katayama [120, 121] is a generalization of LQI optimal control for the case when future changes to the reference signal are available or *previewed*. This is important since the x_r trajectory that achieves a desired *zmp* trajectory with minimal jerk should accelerate earlier than the change in foot support (Figure B.1). PC has two further advantages. It gives an optimal trajectory for a desired metric and it can be evaluated in constant time after the pre-computation of gain matrices.

In order to apply PC we require a discrete linear dynamical system, Eq. B.0.1, and a quadratic optimization criterion, Eq. B.0.2. For biped applications $\mathbf{x}$ is the position, velocity and acceleration of x_r, $\mathbf{y}$ is the *zmp* and $\mathbf{u}$ is the control input. The goal is to

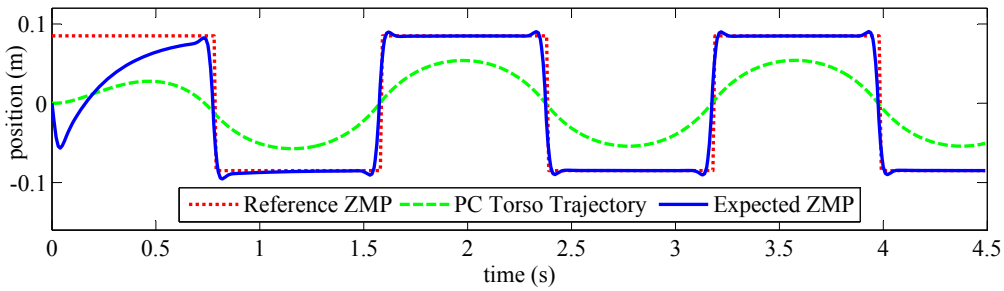


Figure B.1: Computed torso trajectory for five steps.

find a sequence of controls that minimizes J, a relative weighting of tracking error, $\mathbf{e}(t) = \mathbf{y}(t) - \mathbf{y}_d(t)$, change to state $\Delta\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}(t-1)$ and control $\Delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}(t-1)$. In the following, assume that A,B and C have dimension $n \times n, n \times r$ and $p \times n$ respectively.

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) \qquad \text{(B.0.1)} \\ J &= \sum_{i=t}^{\infty} \mathbf{e}(i)Q_e\mathbf{e}(i) + \Delta\mathbf{x}Q_x\Delta\mathbf{x}(i) + \Delta\mathbf{u}^T(i)R\Delta\mathbf{u}(i) \qquad \text{(B.0.2)} \end{aligned}$$

Preview control assumes that N future values of $\mathbf{y}_d^t$ are known and that $\mathbf{y}_d^t(i) = \mathbf{y}_d^t(N)$ for $i > N$. First, we transform this problem into LQI form and present the reduced solution. Next, we address the specifics of biped control.

B.1 Transformation to LQI

In order to express the optimization problem in standard LQI form, Katayama creates an augmented state vector given in Eq. B.1.1. The state consists of observation error, state change and expected reference changes, $\Delta\mathbf{y}_d(i) = \mathbf{y}_d(i) - \mathbf{y}_d(i-1)$. For N known future reference changes, $\bar{\mathbf{x}}$ is a state vector of length $(p + n + pN)$:

$$\bar{\mathbf{x}}(t) = \begin{bmatrix} \mathbf{e}^T(t) & \Delta\mathbf{x}^T(t) & \Delta\mathbf{y}_d^T(t+1) & \dots & \Delta\mathbf{y}_d^T(t+N) \end{bmatrix}^T \qquad \text{(B.1.1)}$$

Given the linear relationship in Eq. B.0.1, the augmented state update is also linear. Individually, the difference equations for the state components are given in Eq. B.1.2.

$$\begin{aligned}\mathbf{e}(t+1) &= \mathbf{e}(t) + \mathbf{CA}\Delta x(t) + \mathbf{CB}\Delta u(t) - \Delta \mathbf{y}_d(t+1) \\ \Delta \mathbf{x}(t+1) &= \mathbf{A}\Delta \mathbf{x}(t) + \mathbf{B}\Delta \mathbf{u}(t)\end{aligned} \tag{B.1.2}$$

For $i = 1, \ldots, N-1$, $\Delta \mathbf{y}_d(t+i)$ in $\bar{\mathbf{x}}(t+1)$ equals its counterpart in $\bar{\mathbf{x}}(t)$. Since the reference stays constant after N steps, $\Delta \mathbf{y}_d(t+N) = \mathbf{0}$. Given $p \times p$ identity matrix, $\mathbf{I}$, we merge the updates and restate the optimization in terms of the new state vector.

$$\bar{\mathbf{x}}(t+1) = \left[\begin{array}{cc|cccc} \mathbf{I} & \mathbf{CA} & -\mathbf{I} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} \\ \hline \mathbf{0} & & \mathbf{0} & \mathbf{I} & \ldots & \mathbf{0} \\ \vdots & & & \ddots & \ddots & \mathbf{0} \\ & \vdots & & & \mathbf{0} & \mathbf{I} \\ & \mathbf{0} & & & & \mathbf{0} \end{array}\right] \bar{\mathbf{x}}(t) + \left[\begin{array}{c} \mathbf{CB} \\ \mathbf{B} \\ \vdots \\ \mathbf{0} \\ \vdots \end{array}\right] \Delta \mathbf{u}(t) \tag{B.1.3}$$

$$J = \sum_{i=t}^{\infty} \bar{\mathbf{x}}^T(i) \begin{bmatrix} Q_e & \cdots & \mathbf{0} \\ \vdots & Q_x & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \bar{\mathbf{x}}(i) + \Delta \mathbf{u}^T(i) R \Delta \mathbf{u}(i) \tag{B.1.4}$$

The system and criterion define a strict LQI problem. As proved by Katayama [121], the optimal incremental control $\Delta u^*(k)$ has the form:

$$\Delta u^*(t) = -G_1 \mathbf{e}(t) - G_2 \Delta \mathbf{x}(t) - \sum_{i=1}^{N} G_3(i) \Delta \mathbf{y}_d(t+i) \tag{B.1.5}$$

To find the gains, G_1,G_2 and G_3, we solve the Riccati equation in Eq. B.1.6. For simplicity we use additional matrices defined in Eq. B.1.7.

$$\mathbf{K} = \bar{\mathbf{A}}^T \mathbf{K} \bar{\mathbf{A}} - \bar{\mathbf{A}} \mathbf{K} \bar{\mathbf{B}} [R + \bar{\mathbf{B}}^T \mathbf{K} \bar{\mathbf{B}}]^{-1} \bar{\mathbf{B}}^T \mathbf{K} \bar{\mathbf{A}} + \bar{\mathbf{Q}} \tag{B.1.6}$$

$$\bar{\mathbf{B}} = \begin{bmatrix} \mathbf{CB} \\ \mathbf{B} \end{bmatrix} \bar{\mathbf{I}} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \bar{\mathbf{F}} = \begin{bmatrix} \mathbf{CA} \\ \mathbf{A} \end{bmatrix} \bar{\mathbf{Q}} = \begin{bmatrix} Q_e & \mathbf{0} \\ \mathbf{0} & Q_x \end{bmatrix} \bar{\mathbf{A}} = \begin{bmatrix} \bar{\mathbf{I}} & \bar{\mathbf{F}} \end{bmatrix} \tag{B.1.7}$$

Let $\mathbf{W} = [R + \bar{\mathbf{B}}^T \mathbf{K} \bar{\mathbf{B}}]^{-1} \bar{\mathbf{B}}^T$. The gains are computed as follows:

$$\begin{aligned} G_1 &= \mathbf{WK}\bar{\mathbf{I}} \qquad & G_3(i) &= -\mathbf{W}([\bar{\mathbf{A}} - \bar{\mathbf{B}}\mathbf{WK}\bar{\mathbf{A}}]^T)^{i-1}\mathbf{K}\bar{\mathbf{I}} \\ G_2 &= \mathbf{WK}\bar{\mathbf{F}} & & i = 1, \ldots, N \end{aligned} \tag{B.1.8}$$

To find the optimal state trajectory we perform simple discrete integration over the state update equations by applying the incremental control.

B.2 Biped Application

The application of preview control to biped locomotion follows directly from the linear model presented in Eq. 4.2.2. Let $\mathbf{x} = [\ x_r\ \dot{x}_r\ \ddot{x}_r\ ]$ and $\mathbf{y} = [\ zmp\ ]$ or $[\ \beta\ ]$ when considering external forces. Also assume that we control the third derivative, jerk, of x_r ($\mathbf{u} = [\ \dddot{x}_r\ ]$) [46]. In accordance with (B.0.1) we represent the model as a discrete state space system with a sampling time $T = .01s$.

$$\begin{aligned} \mathbf{x}(t+1) &= \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} \mathbf{u}(t) \\ \mathbf{y}(t) &= \begin{bmatrix} 1 & 0 & -z_r/g \end{bmatrix} \mathbf{x}(t) \end{aligned} \tag{B.2.1}$$

Regarding the optimization criterion, (B.0.2), we empirically establish $Q_e = 10^7$ and $Q_u = 10$. Q_x is not used since state change is solely dependent on control change during trajectory generation. In order to find the optimal gains we simply iterate the Ricatti equation until convergence. This is sufficient since the gains are determined in advance and are not recomputed online.

C

Coordinate Transformations

The use of any coordinate system requires us to derive translations of displacements. Eq. C.0.1 gives the mapping from a rotation matrix to fixed axis coordinates. For R_{ab} representing the value at row a, column b of R:

$$\begin{aligned} \psi &= atan2(R_{32}, R_{33}) \\ \theta &= -asin(R_{31}) \\ \phi &= atan2(R_{21}, R_{11}) \end{aligned} \tag{C.0.1}$$

In order to apply TS or FR, we also need to map velocities in workspace to velocities in task space. The angular velocity vector ω is found by a summation of fixed axis velocities in a common frame:

$$\omega = \begin{bmatrix} 0 \\ 0 \\ \dot{\phi} \end{bmatrix} + R_z(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_z(\phi) R_y(\theta) \begin{bmatrix} \dot{\psi} \\ 0 \\ 0 \end{bmatrix} \tag{C.0.2}$$

This relationship is rearranged as a single matrix, $\mathbf{E}_{\omega}^{-1}(q)$:

$$\omega = \mathbf{E}_{\omega}^{-1}(q) \begin{bmatrix} \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}^T \tag{C.0.3}$$

Inverting $\mathbf{E}_{\omega}^{-1}(q)$ we find $\mathbf{E}_{\omega}(q)$, a matrix that maps angular velocities to fixed axis velocities. Adjoining the identity matrix for linear velocities Eq. C.0.4 specifies $\mathbf{E}(q)$.

$$\mathbf{E}_{rpy}(q) = \begin{bmatrix} \mathbf{I_{3\times 3}} & \cdots & 0 & \cdots \\ \vdots & c_\phi/c_\theta & s_\phi/c_\theta & 0 \\ 0 & -s_\phi & c_\phi & 0 \\ \vdots & c_\phi s_\theta/c_\theta & s_\phi s_\theta/c_\theta & 1 \end{bmatrix} \tag{C.0.4}$$

Bibliography

[1] P. Buerhaus, K. Donelan, B. Ulrich, L. Norman, and B. Dittus. State of the Registered Nurse Workforce in the United States. *Nurs Econ*, 24(1):6–12, 2006.

[2] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot HRP-2. *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 2004.

[3] O. Khatib, K. Yokoi, O. Brock, K. Chang, and A. Casal. Robots in Human Environments: Basic Autonomous Capabilities. *The International Journal of Robotics Research*, 18(7):684, 1999.

[4] S. Clifford, D. Haddox, D. Ekdahl, and JS Tulenko. Real-time control of the ANDROS mobile robot. *Transactions of the American Nuclear Society*, 70(35), 1994.

[5] H.H. Kwee. Integrated control of MANUS manipulator and wheelchair enhanced by environmental docking. *Robotica*, 16(05):491–498, 1998.

[6] Nils J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

[7] T. Lozano-Perez and P.H. Winston. LAMA: A Language for Automatic Mechanical Assembly. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.

[8] G. J. Fahlman. A planning system for robot construction tasks. Technical Report 283, MIT: Artificial Intelligence Laboratory Series, May 1973.

[9] G. J. Sussman. The findspace problem. Technical Report 286, MIT: Artificial Intelligence Laboratory Series, March 1973.

[10] T. Lozano-Perez. Spatial planning: a configuration space approach. *IEEE Trans. Comput.*, pages 108–120, 1983.

[11] H. Inoue. Force feedback in precise assembly tasks. Technical Report 308, MIT: Artificial Intelligence Laboratory Series, Aug 1974.

[12] Tomas Lozano-Perez, Matthew Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[13] M. H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *Int. J. of Computational Geometry and Applications*, 9:371–418, 1999.

[14] R. Wilson. *On Geometric Assembly Planning*. PhD thesis, Department of Computer Science, Stanford University, 1992.

[15] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. Journal of Robotics Research*, 15(6):533–556, 1996.

[16] M.T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[17] R. Alami, J.P. Laumond, and T. Sim'eon. Two manipulation planning algorithms. In *Workshop on the Algorithmic Foundations of Robotics*, 1994.

[18] T. Simeon, J. Cortes, A. Sahbani, and J.P. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 2002.

[19] M.A. Erdmann. On Motion Planning with Uncertainty. 1984.

[20] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[21] O. Ben-Shahar and E. Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Trans. on Robotics and Automation*, 14(4):549–565, 1998.

[22] A. Junghanns and J. Schaffer. Sokoban: A challenging single-agent search problem. In *IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*, pages 27–36, 1997.

[23] Thomas Chadzelek, Jens Eckstein, and Elmar Schömer. Heuristic motion planning with many degrees of freedom. Technical Report FB14-95-08, Univerität des Saarlandes, August 1995.

[24] J. Ota. Rearrangement of multiple movable objects. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1962–1967, 2004.

[25] P.C.Chen and Y.K.Hwang. Pracitcal path planning among movable obstacles. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 444–449, 1991.

[26] J.C. Latombe. *Robot Motion Planning*. Springer, 1991.

[27] Andrew T. Miller. *GraspIt: A Versatile Simulator for Robotic Grasping*. PhD thesis, Dept. of Computer Science, Columbia University, 2001.

[28] Michael Erdmann. An exploration of nonprehensile two-palm manipulation. *International Journal of Robotics Research*, 17(5), 1998.

[29] G. Morris and L. Haynes. Robotic assembly by constraints. *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, 4, 1987.

[30] JD Morrow and PK Khosla. Manipulation task primitives for composing robot skills. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, 4, 1997.

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

[32] G. Wilfong. Motion panning in the presence of movable obstacles. In *Proc. ACM Symp. Computat. Geometry*, pages 279–288, 1988.

[33] E. Demaine and et. al. Pushpush and push-1 are np-complete. Technical Report 064, Smith, 1999.

[34] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1):219–251, 2001.

[35] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. *AIPS98 Workshop on Planning as Combinatorial Search*, pages 58–60, 1998.

[36] J.C. Culberson and J. Schaeffer. Searching with pattern databases. *Lecture Notes in Computer Science*, 981:101–112, 2001.

[37] R.E. Korf. Finding optimal solutions to Rubiks Cube using pattern databases. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 700–705, 1997.

[38] L. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4), 1996.

[39] S.M. LaValle and J.J. Kuffner. Rapidly exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[40] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, 3, 1997.

[41] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[42] K. Inoue, H. Yoshida, T. Arai, and Y. Mae. Mobile manipulation of humanoids: Real-time control based on manipulability and stabilty. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 2217–2222, 2000.

[43] J. Kuffner. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1), 2002.

[44] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa. Pushing manipulation by humanoid considering two-kinds of zmps. In *IEEE Int. Conf. on Robotics and Automation*, pages 1627–1632, 2003.

[45] K. Harada, S. Kajita, F. Kanehiro, K.Fujiwara, K. Kaneko, K.Yokoi, and H. Hirukawa. Real-time planning of humanoid robot's gait for force controlled manipulation. In *IEEE Int. Conf. on Robotics and Automation*, pages 616–622, 2004.

[46] Shuuji Kajita and et. al. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE Int. Conf. on Robotics and Automation*, pages 1620–1626, 2003.

[47] T. Takubo, K. Inoue, and T. Arai. Pushing an object considering the hand reflect forces by humanoid robot in dynamic walking. In *IEEE Int. Conf. on Robotics and Automation*, pages 1718–1723, 2005.

[48] K. Nishiwaki, W-K. Yoon, and S. Kagami. Motion control system that realizes physical interaction between robot's hands and environment during walk. In *IEEE Int. Conf. on Humanoid Robotics*, 2006.

[49] K. Nishiwaki and S. Kagami. High frequency walking pattern generation based on preview control of zmp. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'06)*, 2006.

[50] E. Yoshida, I. Belousov, Claudia Esteves, and J-P. Laumond. Humanoid motion planning for dynamic tasks. In *IEEE Int. Conf. on Humanoid Robotics (Humanoids'05)*, 2005.

[51] E. Yoshida, C. Esteves, T. Sakaguchi, J-P. Laumond, and K. Yokoi. Smooth collision avoidance: Practical issues in dynamic humanoid motion. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[52] E. Krotkov. Robotic perception of material. In *IJCAI*, pages 88–95, 1995.

[53] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through interaction - initial steps towards artificial cognition. In *IEEE Int. Conf. on Robotics and Automation*, pages 3140–3145, 2005.

[54] A. Stoytchev. Behavior-grounded representation of tool affordances. In *IEEE Int. Conf. on Robotics and Automation*, pages 3060–3065, 2005.

[55] A. Christiansen, T. M. Mitchell, and M. T. Mason. Learning reliable manipulation strategies without initial physical models. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1990.

[56] C.H. An, C.G. Atkeson, and J.M. Hollerbach. *Model-Based Control of a Robot Manipulator*. MIT Press, 1988.

[57] C. Canudas de Wit, P. Nol, A. Aubin, and B. Brogliato. Adaptive friction compensation in robot manipulators: low velocities.

[58] H. Olsson, KJ Astrom, CC. de Wit, M. Gafvert, and P. Lischinsky. Friction models and friction compensation.

[59] Stefan Schaal Chris Atkeson, Andrew Moore. Locally weighted learning. *AI Review*, 11:11–73, April 1997.

[60] Andrew Moore, C. G. Atkeson, and S. A. Schaal. Locally weighted learning for control. *AI Review*, 11:75–113, 1997.

[61] M. Vukobratovic, B. Borovac, D. Surla, and D. Stokic. *Biped Locomotion: Dynamics, Stability, Control and Application*. Springer, 1990.

[62] M. Vukobratovic and B. Borovac. Zero-moment point-thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.

[63] D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man Machine Systems*, 10:47–53, 1969.

[64] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, K. Fujimura, H.R.D.C. Ltd, and J. Saitama. The intelligent ASIMO: system overview and integration. *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, 3, 2002.

[65] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami. Planning biped navigation strategies in complex environments. In *2003 International Conference on Humanoid Robots*, 2003.

[66] J. Gutmann, M. Fukuchi, and M. Fujita. Real-time path planning for humanoid robot navigation. In *International Joint Conference on Artificial Intelligence*, 2005.

[67] E. Yoshida, P. Blazevic, and V. Hugel. Pivoting manipulation of a large object. In *IEEE Int. Conf. on Robotics and Automation*, pages 1052–1057, 2005.

[68] R. Brooks, L. Aryananda, A. Edsinger, P. Fitzpatrick, C. Kemp, U.M. OReilly, E. Torres-Jara, P. Varshavskaya, and J. Weber. Sensing and Manipulating Built-for-Human Environments. *International Journal of Humanoid Robotics*, 1(1):1–28, 2004.

[69] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. DARPA Image Understanding Workshop*, 121:130, 1981.

[70] R.M. Haralick and L.G. Shapiro. *Computer and Robot Vision*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1992.

[71] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[72] K. Dorfmuller. Robust tracking for augmented reality using retroreflective markers. *Computers and Graphics*, 23(6):795–800, 1999.

[73] Y. Argotti, L. Davis, V. Outters, and J. Rolland. Dynamic superimposition of synthetic objects on rigid and simple-deformable real objects. *Computers and Graphics*, 26(6):919, 2002.

[74] A. State, G. Hirota, D.T. Chen, W.F. Garrett, and M.A. Livingston. Superior augmented reality registration by integrating landmark tracking and magnetic tracking. In *In Proc. SIGGRAPH'96*, page 429, 1996.

[75] M. Stilman, P. Michel, J. Chestnutt, K. Nishiwaki, S. Kagami, and J. Kuffner. Augmented reality for robot development and experimentation. Technical Report CMU-RI-TR-05-55, Robotics Institute, Carnegie Mellon University, November 2005.

[76] J. Bentley, G.M. Faust, and F. Preparata. Approximation algorithms for convex hulls. *Comm. of the ACM*, 25(1):64–68, 1982.

[77] JJ Kuffner Jr, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *International Conference on Intelligent Robots and Systems*, page 500, 2001.

[78] A. Stentz. Optimal and efficient path planning for partially-knownenvironments. *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317, 1994.

[79] M. Jeannerod, M.A. Arbib, G. Rizzolatti, and H. Sakata. Grasping objects: the cortical mechanisms of visuomotor transformation. *Trends Neurosci.*, 18:314–320, 1995.

[80] D. Nieuwenhuisen and M. Overmars F. van der Stappen. An effective framework for path planning amidst movable obstacles. In *Workshop on the Algorithmic Foundations of Robotics*, 2006.

[81] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. In *IEEE Int. Conf. on Robotics and Automation*, pages 1419–1424, Apr. 7-10 1986.

[82] M. A Ganter. *Dynamic Collision Det. using Kinematics and Solid Modeling Techniques(Mechanical Engineering)*. PhD thesis, University of Wisconsin, 1985.

[83] A. Foisy and V. Hayward. A safe swept-volume approach to collision detection. In *Robotics Research, Sixth International Symposium*, 1994.

[84] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue. Environment manipulation planner for humaonid robots using task graph that generates action sequence. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'04)*, pages 1174–1179, 2004.

[85] M. Stilman, J. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'07)*, 2007.

[86] J. Schamburek. Diploma thesis: Manipulation planning among movable obstacles. Technical report, University of Karlsruhe (TH), 2007.

[87] K. Gupta J. Ahuactzin and E. Mazer. Manipulation planning for redundant robots: A practical approach. *International Journal of Robotics Research*, 17(7), 1998.

[88] Y. Hirano, K. Kitahama, and S. Yoshizawa. Image-based object recognition and dexterous hand/arm motion planning using rrts for grasping in cluttered scene. pages 3981–3986, 2005.

[89] M. T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Trans. on Systems, Man, and Cybernetics,*, 11(6), 1981.

[90] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *International Journal of Robotics Research*, 3(1), 1987.

[91] J. Ahuactzin and K. Gupta. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Trans. on Robotics and Automation*, 15:653–669, 1999.

[92] Z. Guo and T. Hsia. Joint trajectory generation for redundant robots in an environment with obstacles. In *IEEE Int. Conf. on Robotics and Automation*, pages 157–162, 1990.

[93] G. Oriolo, M. Ottavi, and M. Vendittelli. Probabilistic motion planning for redundant robots along given end-effector paths. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2002.

[94] G. Oriolo and C. Mongillo. Motion planning for mobile manipulators along given end-effector paths. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'05)*, 2005.

[95] D. P. Martin, J. Baillieul, and J. M. Hollerbach. Resolution of kienmatic redundancy using opetimization techniques. *IEEE Trans. on Robotics and Automation*, 5(4):529–533, 1989.

[96] B. Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3:201–212, 1990.

[97] S. Seereeram and J. T. Wen. A global approach to path planning for redundant manipulators. *IEEE Trans. on Robotics and Automation*, 11(1), 1995.

[98] A. McLean and S. Cameron. The virtual springs method: Path planning and collision avoidance for redundant manipulators. 15, 1996.

[99] Oliver Brock, Oussama Khatib, and Sriram Viji. Task-consistent obstacle avoidance and motion behavior for mobile manipulation. In *IEEE Int'l Conf. on Robotics and Automation*, 2002.

[100] O. Khatib, L. Sentis, J. Park, and J. Warren. Whole body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, pages 29–44, 2004.

[101] S.M.LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[102] S. M. LaValle, J. Yakey, and L. E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *In Proc. IEEE Int'l Conf. on Robotics and Automation*, 1999.

[103] L. Han and N. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2000.

[104] J. Cortes T. Simeon and J.P. Laumond. A random loop generator for planning the motions of closed kinematic chains with prm methods. In *IEEE Int. Conf. on Robotics and Automation*, 2002.

[105] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'01)*, pages 692–698, 2001.

[106] Z. Yao and K. Gupta. Path planning with general end-effector constraints: Using task space to guide configuration space search. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2005.

[107] L. Sciaviccio and B. Siciliano. *Modeling and control of robot manipulators.* McGraw-Hill Co., 1996.

[108] T. N. E. Greville. Some applications of the pseudoinverse of a matrix. *SIAM Review*, 2:15–22, 1960.

[109] J. Yakey, S.M. LaValle, and L. E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(7), 2001.

[110] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'02)*, 2002.

[111] T. Yoshikawa. Manipulability of robotic mechanisms. *Int. Journal of Robotics Research*, 4:3–9, 1985.

[112] H. Hanafusa, T. Yoshikawa, and Y. Nakamura. Analaysis and control of articulated robot with redundancy. In *IFAC, 8th Triennal World Congress*, volume 4, pages 1927–1932, 1981.

[113] R. Boulic P. Baerlocher. Task-priority formulations for the kinematic control of highly redundant atriculated structures. In *Int. Conf. on Intelligent Robots and Systems*, 1998.

[114] J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greenberg. Real-time robot motion planning using rasterizing computer. *Computer Graphics, ACM*, 24(4):327–335, 90.

[115] S.A. Setiawan, J. Yamaguchi, and T. SHH. Physical interaction between human and a bipedal humanoid robot. *IEEE Conference on Robotics and Automation*, 1:361–367, 1999.

[116] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa. The 3D linear inverted pendulum mode: a simple modeling for a bipedwalking pattern generation. *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 1, 2001.

[117] K. Nagasaka, H. Inoue, and M. Inaba. Dynamic walking pattern generation for a humanoid robot based on optimal gradient method. *PROC IEEE INT CONF SYST MAN CYBERN*, 6, 1999.

[118] K. Nishiwaki, T. Sugihara, S. Kagami, M. Inaba, and H. Inoue. Online mixture and connection of basic motions for humanoid walking control by footprint specification. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 4, 2001.

[119] K. Yokoi, F. Kanehiro, K. Kaneko, K. Fujiwara, S. Kajita, and H. Hirukawa. A Honda Humanoid Robot Controlled by AIST Software. *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, pages 259–264, 2001.

[120] M. Tomizuka and D. Rosenthal. On the optimal digital state feedback controller with integral and preview actions. *Transactions of ASME, Journal of Dynamic Systems, Measurement and Control*, 101, 1979.

[121] T. Katayama, T. Ohki, T. Inoue, and T. Kato. Design of an optimal con-troller for a discrete time system subject to previewable demand. *Int. Journal of Control*, 41(3), 1985.