Chapter 8
Autonomous Manipulation of Movable Obstacles

Mike Stilman

Abstract In this chapter we describe recent progress towards autonomous manipulation of environment objects. Many tasks, such as nursing home assistance, construction or search and rescue, require the robot to not only avoid obstacles but also move them out if its way to make space for reaching the goal. We present algorithms that decide which objects should be moved, where to move them and how to move them. Finally, we introduce a complete system that takes into account humanoid balance, joint limits and fullbody constraints to accomplish environment interaction.

8.1 Introduction

Humanoid robots would be much more useful if they could move obstacles out of the way. In this section, we address the challenges of autonomous navigation among movable obstacles (NAMO), present algorithms and evaluate their effectiveness in simulation and present a successful implementation of NAMO on a humanoid robot. Traditional motion planning searches for collision freepaths from a start to a goal. However, realworld 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 environment, decides which objects must be moved and clears a way to further progress. Human safety is one of many motivations for

Mike Stilman
Georgia Tech, Atlanta, GA, USA e-mail: mstilman@cc.gatech.edu

Figure 8.1 Planned solution to a *navigation* problem that manipulates an environment object

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 8.1, even a chair in front of a table challenges the robot to NAMO.

8.1.1 Planning Challenges

We begin our analysis of the NAMO domain as an instance of geometric motion planning [2]. 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:

- $O_F = \{F_1, \ldots, F_f\}$ - a set of polyhedral *Fixed Obstacles* that must be avoided.
- $O_M = \{O_1, \ldots, O_m\}$ - 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.

8.1.2 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:

$$Navigate : (W^t, \tau(r^t, r^{t+1})) \rightarrow W^{t+1}. \tag{8.1}$$

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 $G_i \in G(O_i)$. The set $G(O_i)$ consists of relative transformations between the robot end-effector and the object that constitute contact.

$$Manipulate : (W^t, O_i, G_i, \tau(r^t, r^{t+1})) \rightarrow W^{t+1}. \tag{8.2}$$

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 8.1.3.

The two action descriptions point to a general formulation for interacting with environment objects. The robot iterates a twostep 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.

8.1.3 Action Spaces

In Section 8.1.2 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 Sections 8.2 and 8.3. 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 [3]. 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 [4]. Such grasps may either be specified by the user or computed automatically [5]. 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 [4]. *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 [6, 7]. 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.[6]

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 interactions between a robot end-effector and an object [8, 9]. 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 [10] 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 of significant 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.

8.1.4 Complexity of Search

In contrast to a mobile robot, the branching factor of search for humanoid robots 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 [11]. More recent work demonstrated NP-completeness results for trivial problems where square blocks can translated on a planar grid [12].

In order to better understand the source of complexity, consider a simplified grid world that contains a robot and m obstacles as shown in Figure 8.2. Suppose we attempt a complete search over the robot's action space. Let the configuration of the robot base be represented by $\mathscr{C}_R = (x, y)$, with resolution n in each direction. The generic size of the robot $\mathscr{C}$-space $|\mathscr{C}_R| = O(n^2)$. Analogously, for each object $|\mathscr{O}_i| = O(n^2)$. The full space of possible environment configurations is the product of these subspaces, $\mathscr{C}_R \times \mathscr{O}_1 \times \mathscr{O}_2 \times \ldots \times \mathscr{O}_N$, 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 8.2, 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 nine: four directions of motion,

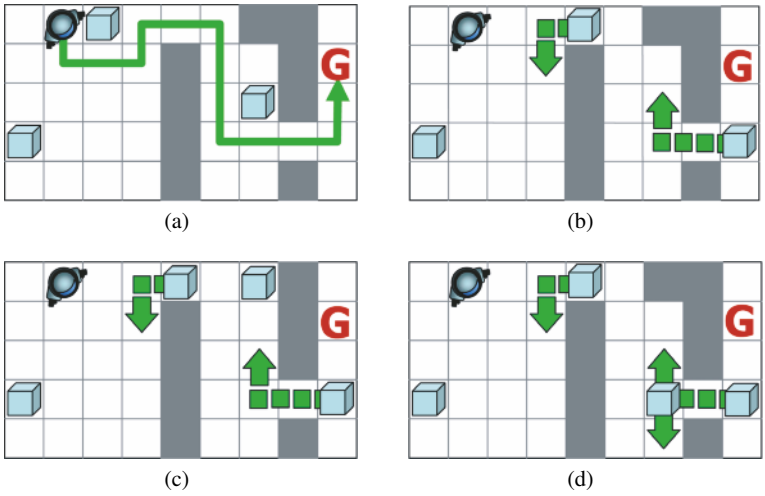


Figure 8.2 Simple NAMO problems on a planar grid: (a) N: 3, Moved: 0, States Searched: 4,840; (b) N: 3, Moved: 2, States Searched: 14,637; (c) N: 4, Moved: 2, States Searched: 48,264; (d) N: 4, Moved: 3, States Searched: 32,258

four 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 8.2(a), although no obstacles are moved, the search examines 4840 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 8.2(c) may look more like Figure 8.2(b) than Figure 8.2(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 [13] applied the *Blackbox* [14] 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 [15, 16, 13]. 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 grid world NAMO. However, our interest is in NAMO

as a practical robot domain. Arbitrary object geometry and higher granularity action sets make pre-computation infeasible for any 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 [17, 18]. These methods are most effective in *expansive* spaces where it is easy to sample points that significantly expand the search tree [19]. NAMO problems typically have numerous narrow passages in the state space. In Figure 8.2, among thousands of explored actions only one or two object grasps and translations make progress to the goal. Section 8.2 will discuss the narrow passages that exist in NAMO planning and use them to guide search.

8.2 NAMO Planning

8.2.1 Overview

In Section 8.1 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.

8.2.2 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 [20]. Let $\mathscr{C}_W$ 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 $\mathscr{C}_W$:

$$\mathscr{C}_R(W^t) = (\{r\}, q_1^t, q_2^t, \ldots, q_m^t). \tag{8.3}$$

While $\mathscr{C}_R$ includes all possible configurations for the robot, some 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:

$$A(q) = \{x \in \mathrm{R}^k | x \text{ is a point of object } A \text{ in configuration } q\}. \tag{8.4}$$

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

$$p \in \mathscr{X}_B(A(q)) \Rightarrow B(p) \cap A(q) \neq \emptyset \Rightarrow q \in \mathscr{X}_A(B(p)). \tag{8.5}$$

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

The free space of a robot, $\mathscr{C}_A^{free}(W^t)$, is the set of configurations where the object is not in collision with fixed or movable obstacles. Eq. 8.6 defines $\mathscr{C}_R^{free}(W^t)$ and $\mathscr{C}_{O_i}^{free}(W^t)$ as sets of collision free configuration for the robot and movable objects. Figure 8.3(a) shows the free configuration space $\mathscr{C}_R^{free}(W^t)$ for a circular robot.

$$\mathscr{C}_R^{free}(W^t) = \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_i \overline{\mathscr{X}_R^{O_i}(W^t)} \qquad \mathscr{C}_{O_i}^{free}(W^t) = \bigcap_k \overline{\mathscr{X}_{O_i}(F_k)} \bigcap_{O_j \neq O_i} \overline{\mathscr{X}_{O_i}^{O_j}(W^t)} \tag{8.6}$$

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

$$\tau(s) \in \mathscr{C}_R^{free}(W^t) \qquad \forall s \in [0,1]. \tag{8.7}$$

Equations 8.8 – 8.12 validate a *Manipulate* operator. In addition to satisfying collision free motion manipulation must end with the object in a statically stable *placement* $\mathscr{C}_{O_i}^{place}(W^t) \subseteq \mathscr{C}_{O_i}^{free}(W^t)$ (Equation 8.11). Equation Eq. 8.12 ensures that the robot does not collide with obstacle O_i. In our 2D examples, we assume gravity is orthogonal to the object plane and hence $\mathscr{C}_{O_i}^{place}(W^t) = \mathscr{C}_{O_i}^{free}(W^t)$.

$$\tau(s) \in \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_{j \neq i} \overline{\mathscr{X}_R^{O_j}(W^t)} \quad \forall s \in [0,1] \tag{8.8}$$

$$\tau_{O_i}(s) \in \mathscr{C}_{O_i}^{free}(W^t) \quad \forall s \in [0,1] \tag{8.9}$$

$$\tau_{O_i}(0) = q_i^t \tag{8.10}$$

$$\tau_{O_i}(1) \in \mathscr{C}_{O_i}^{place}(W^t) \tag{8.11}$$

$$R(\tau(s)) \cap O_i(\tau_{O_i}(s)) = \emptyset \quad \forall s \in [0,1]. \tag{8.12}$$

8.2.3 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 8.3(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. Equation 8.13 generalizes this relationship to all robot configurations that are accessible from r^t in one step of *Navigate*:

$$\mathscr{C}_R^{acc}(W^t) = \{r_j \in \mathscr{C}_R^{free}(W^t) | \text{ exists } \tau(r^t, r_j) \text{ s.t. } \forall s(\tau[s] \in \mathscr{C}_R^{free}(W^t)\}. \quad (8.13)$$

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

$$\forall r_i, r_j \in \mathscr{C}_R^{acc}(W^t) \text{ exists } \tau(r_i, r_j) \text{ s.t. } \forall s(\tau[s] \in \mathscr{C}_R^{acc}(W^t)). \quad (8.14)$$

The space of accessible configurations is lightly shaded in Figure 8.3(a). We can compute configurations that belong to this set using grid-based wavefront propagation [21].

Accessible configurations are feasible goals for navigation. Likewise, contact configurations are feasible starting states for manipulation. Any class of *Manipulate* operators in Section 8.1.3 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 $G(O_i)$ 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):

$$\mathscr{C}_R^{cont}(O_i, W^t) = IK(G(O_i), W^t). \quad (8.15)$$

In Figure 8.3(b) the table can be grasped at any point on the perimeter. The shaded area represents $\mathscr{C}_R^{cont}(Table_1, W^0)$ as a set of feasible positions for the robot base that make it possible to grasp the object. Figure 8.3(c) illustrates the intersubsection of $\mathscr{C}_R^{acc}$ and $\mathscr{C}_R^{cont}$. These configurations are object contacts that are accessible to the robot in W^t.

$$\mathscr{C}_R^{AC}(O_i, W^t) = \mathscr{C}_R^{acc}(W^t) \bigcap \mathscr{C}_R^{cont}(O_i, W^t). \quad (8.16)$$

The set $\mathscr{C}_R^{AC}(O_i, W^t)$ represents useful goals for the *Navigate* operator in W^t. There are two cases. When $\mathscr{C}_R^{acc}(W^t)$ contains the goal state, NAMO reduces to a path plan

(a) (b) (c)

Figure 8.3 Simulated 2D NAMO $\mathscr{C}_R$-space for a circular robot: (a) $A, B \in \mathscr{C}_R^{acc}(W^0)$; (b) $G(O_i) \rightarrow \mathscr{C}_R^{cont}(O_i, W^t)$; (c) $\mathscr{C}_R^{acc}(W^t) \cap \mathscr{C}_R^{cont}(O_i, W^t)$

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

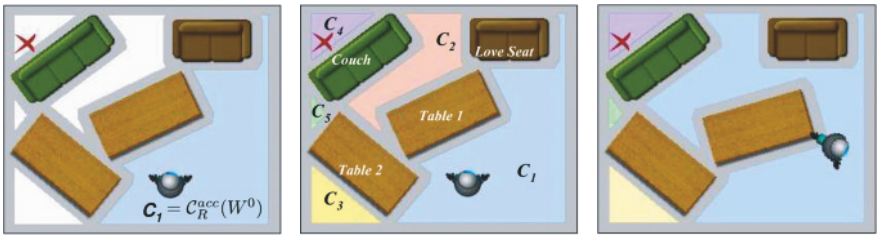
8.2.4 Goals for Manipulation

In Section 8.2.3 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 NAMO domain.

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 8.2.3.

We already observed that $\mathscr{C}_R^{acc} \in \mathscr{C}_R^{free}$ 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 $\mathscr{C}_R^{acc}$. There would also be a set of configurations accessible to the robot. In fact, as shown in Figure 8.4(b), we can partition the robot free space, $\mathscr{C}_R^{free}$, 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 8.4(c). After the action,



(a) (b) (c)

Figure 8.4 NAMO $\mathscr{C}_R$-space partitioned into components: (a) $C_1 = \mathscr{C}_R^{acc}(W^0)$; (b) $\mathscr{C}_R^{free}$ Components; (c) Keyhole solution

configurations in C_2 become valid goals for *Navigate*. The illustration shows part of the solution to a *keyhole* in this NAMO problem.

Definition 8.1. 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:

$$C_i \cap \mathscr{C}_R^{free}(W^2) \subset \mathscr{C}_R^{acc}(W^2) \qquad \text{and} \qquad C_i \cap \mathscr{C}_R^{free}(W^2) \neq \emptyset. \tag{8.17}$$

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 8.2. 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 Equation 8.17.

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.

8.2.5 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.

8.2.5.1 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 $\mathscr{C}_{free}$. 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 8.3. 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 $C_1 = \mathscr{C}_R^{acc}(W_0)$.
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 $r^f \in \mathscr{C}_R^{free}(W_n)$.

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 $C_G \cap \mathscr{C}_R^{free}(W^n)$. Although this definition allows for arbitrary k, in this section 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 8.4(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: $\mathscr{X}_R^{O_i}(W^{t+1}) \subset (\mathscr{X}_R^{O_i}(W^t) \cup \mathscr{C}_R^{acc}(W^t))$. This implies that any solution to (W^{t-1}, C_t) results in W^t s.t. $C_t \cap \mathscr{C}_R^{free}(W^t) = C_t$.

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 $\mathscr{C}_R$ 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 realworld problems. Generally, $\mathscr{C}_R^{free}$ 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.

8.2.5.2 Local Manipulation Search

Solving 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 $C_2 \subset \mathscr{C}_R^{free}$ 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 8.2.3 to find a discrete or sampled set of contact states for *Manipulate*: $\mathscr{C}_R^{AC}(W^t)$. 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 Equation 8.17 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 operates with 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 (8.2.5.1) 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.

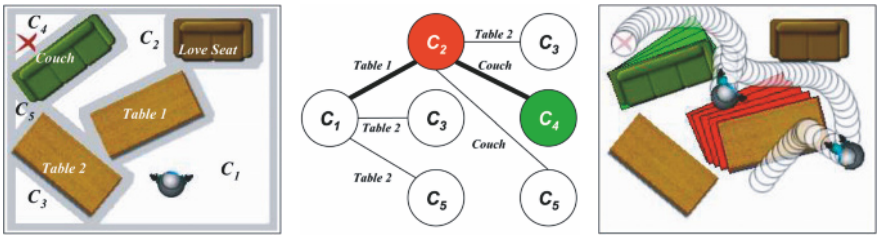
Although MANIP-SEARCH is a simple method, efficient recognition of goal satisfaction is non-trivial. We provide one possible solution in [10].

8.2.5.3 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 $C_i \subset \mathscr{C}_R^{free}$. 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:

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



(a) (b) (c)

Figure 8.5 Construction of FCT structures the solution for L_1 problems: (a) L_1 NAMO Problem; (b) *FCT* with solution path; (c) Solution

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

$$O_N(C_j) = \{O_l | \exists r_j \in C_j, \tau(r^t, r_j) : \forall s(\tau[s] \in \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathscr{X}_R^{O_j}(W^t)})\}. \quad (8.18)$$

3. Remove $C_j \in C_N$. For each $O_l \in O_N(C_j)$ 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 $C_N = \emptyset$.

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 (Figure 8.5).

8.2.5.4 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 8.1. *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,* $\tau[s] \in \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathscr{X}_R^{O_j}(W^T)}$.

Proof. To simplify notation let $A^t = \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_{j \neq l} \overline{\mathscr{X}_R^{O_j}(W^t)}$. 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, $A^t = A^T$ for all t. We now show that every operator path in the *1-solution* satisfies $\tau[s] \in A^T$ 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 $\tau_M[s] \in A^t$ for all s (Equation 8.8). Hence, the path satisfies $\tau_M[s] \in A^T$ for all s.
Navigate$(W^t, \tau_N(r^t, r^{t+1}))$: the operator is valid only if the path satisfies $\tau_N[s] \in \mathscr{C}_R^{free}(W^t)$ for all s (Equation 8.13). Since $\mathscr{C}_R^{free}(W^t) = \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_j \overline{\mathscr{X}_R^{O_j}(W^t)}$, we have $\mathscr{C}_R^{free}(W^t) \subset A^t$. Therefore, $\tau_N[s] \in A^t$ and consequently $\tau_N[s] \in A^T$ for all s.

Furthermore, Equation 8.17, guarantees the existence of $r_j \in C_j \cap \mathscr{C}_R^{free}(W^{T+n})$. Since $C_j \cap \mathscr{C}_R^{free}(W^{T+n}) \subset \mathscr{C}_R^{acc}(W^{T+n})$, there exists $\tau_j(r^{T+n}, r_j)$ such that $\tau_j[s] \in \mathscr{C}_R^{free}(W^{T+n})$ for all s(Equation 8.13). Therefore $\tau_j[s] \in A^{T+n} = A^T$ 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 $\tau[s] \in A^T$ for all s in all subpaths of τ_D, we also have $\tau_D[s] \in A^T$ for all s.

Lemma 8.2. 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 $\mathscr{C}_R^{free}$ 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 $K(W^t, C_i)$ (Equation 8.17). 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 $C_i \cap \mathscr{C}_R^{free}(W_i) \subset \mathscr{C}_R^{acc}(W_i)$.

When (C_{i-1}, W^t) is expanded, Lemma 8.1 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 $O_N(C_i)$. 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 Equation 8.17. 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 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 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 Q have cost $c' > c$. The lowest cost solution would be minimal with respect to MANIP-SEARCH costs.

8.2.5.5 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 of 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 8.2.6 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 $\mathscr{C}_R^{free}$. Lemma 8.1 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 $\mathscr{C}_R$ until they reach the goal and using them to guide the planner in subgoal selection.

8.2.6 Planner Prototype

With the results from the previous section, 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 8.2.5.3). 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.

8.2.6.1 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^I to r^f on a dense regular grid of $\mathscr{C}_R(W^t)$ which plans *through* movable obstacles. RCH keeps a priority queue of grid cells x_i. Each cell is associated with a robot configuration, r_i, the cost of reaching it, $g(x_i)$, and the estimated cost for a solution path that passes through r_i, $f(x_i)$. On each iteration, RCH removes x_i with least $f(x_i)$ from the queue and *expands* it by adding valid adjacent cells x_j with the costs in Equation 8.19. Let $e(r_i, r_j)$ be the transition cost of entering a cell $r_j \in \mathscr{X}_R^{O_j}$ from $r_i \notin \mathscr{X}_R^{O_j}$, 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:

$$\begin{aligned} g(x_j) &= g(x_i) + (1-\alpha) + \alpha e(r_i, r_j), \\ f(x_j) &= g(x_j) + h(x_j, x^f). \end{aligned} \tag{8.19}$$

As shown in Algorithm 8.1, RCH restricts valid transitions. We use *exclusively contained* to refer to r_i contained in only one obstacle: $r_i \in \mathbf{exc}\ \mathscr{X}_R^{O_i} \Rightarrow (O_j \neq O_i$ then $r_i \notin \mathscr{X}_R^{O_j})$.

Definition 8.4. 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 $r_i \in \mathscr{X}_R^{O_i}$ or $r_i \in \mathscr{C}_R^{free}$.

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 $\mathscr{C}_R^{free}$ component. The states searched are triples $x_i = (r_i, O_i, C_j)$. 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 8.3. *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 $(r^I, 0, 0)$ to Q, line 14 expands the children of this state adding all $r_i \in \mathscr{C}_R^{acc}$ 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 $r_j \in \mathscr{X}_R^{O_A}$. Lines 10 and 11 expand any state $(r_i, O_A, 0)$ to $(r_j, O_A, 0)$ where $r_j \in \mathscr{C}_R^{free}$ 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 $(r_i, O_A, 0)$.

Algorithm 8.1 Pseudo-code for RCH.

Algorithm: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 do**
4 $x_1 = (r_1, O_F, C_F) =$ REMOVE-FIRST(Q)
5 **if**$(x_1 \in Closed)$ **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** (x_1)
8 **foreach** $r_2 \in$ ADJACENT(r_1) **do**
9 **if**$(C_F \neq 0)$ ENQUEUE$(Q, (r_2, O_F, C_F))$; **continue**
10 **if**$(O_F \neq 0$ **and** $r_2 \in \mathscr{C}_R^{free})$ ENQUEUE$(Q, (r_2, O_F, 0))$
11 **if**$(O_F \neq 0$ **and** $r_2 \in exc\mathscr{X}_R^{O_F})$ 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** $r_2 \in \mathscr{C}_R^{free})$ENQUEUE$(Q, (r_2, 0, 0))$
15 **if**$(O_F = 0$ **and** $r_2 \in exc\mathscr{X}_R^{O_i})$ENQUEUE$(Q, (r_2, O_i, 0))$
end
end
16 **return** NIL

From Definition 8.4 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 8.4. *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 $\mathscr{C}_R^{free}$ or exclusively in $\mathscr{X}_R^{O_F}$ as added by lines 13, 10 and 11. Finally, every state in P_1 must be in $\mathscr{C}_R^{free}$ since it was added by line 15. Hence the path P satisfies the second criterion of Definition 8.4.

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 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 non-zero C_i path existence will not depend on its value.

8.2.6.2 High-level Planner

Algorithm 8.2 gives the pseudo-code for SELECTCONNECT (SC). The planner makes use of RCH and MANIP-SEARCH, as described in Section 8.2.5.2. 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 $C_i \subset \mathscr{C}_{free}$. 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.

Algorithm 8.2 Pseudo-code for *SelectConnect*.

Algorithm: SelectConnect$(W^t, PrevList, r^f)$

1 $AvoidList \leftarrow \emptyset$
2 **if** $x^f \in \mathscr{C}_R^{acc}(W^t)$ **then**
3 **return** FIND-PATH(W^t, x^f))
end
4 **while** $(O_1, C_1) \leftarrow RCH(W^t, AvoidList, PrevList, r^f) \neq$NIL **do**
5 $(W^{t+2}, \tau_M, c) \leftarrow$ MANIP-SEARCH(W^t, O_1, C_1)
6 **if** $\tau_M \neq$ NIL **then**
7 $FuturePlan \leftarrow$ SELECTCONNECT$(W^{t+2}, PrevList$ **append** $C_1, r^f)$
8 **if** $FuturePlan \neq$ NIL **then**
9 $\tau_N \leftarrow$ FIND-PATH$(W^t, \tau_M[0])$
10 **return** $((\tau_N, \tau_M)$ **append** $FuturePlan)$
end
end
11 $AvoidList$ **append**(O_1, C_1)
end
12 **return** NIL

8.2.6.3 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 8.6(a), we see that $\mathscr{C}_R^{free}$ is disjoint–making this a NAMO problem. Line 4 of SC calls RCH, the heuristic subplanner. 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 8.6(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 8.6(c) shows the minimum cost manipulation

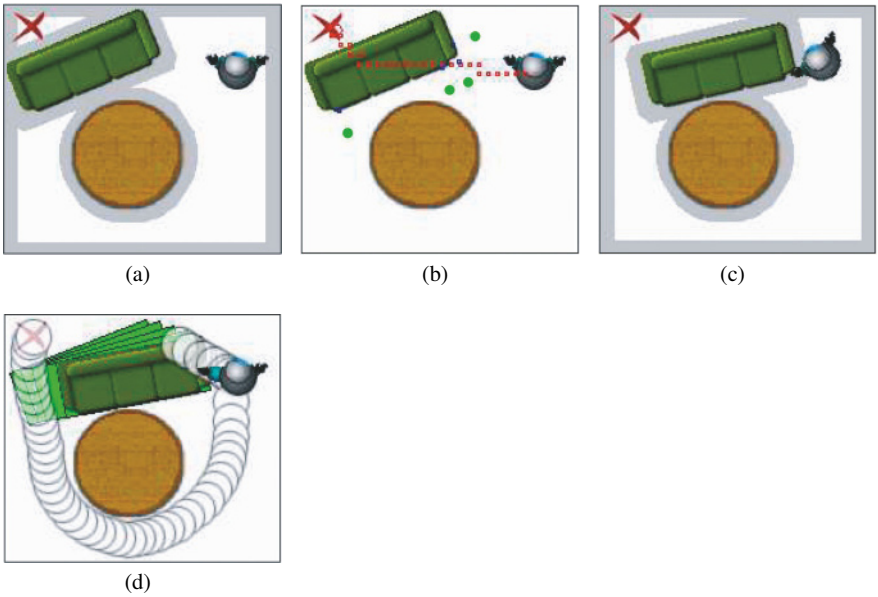


Figure 8.6 Walk-through of an autonomous SELECTCONNECT: (a) Problem; (b) RCH; (c) Keyhole solution; (d) Final plan

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 8.6(d)). The remainder of the pseudo-code iterates this process until the goal is reached and backtracks when a space cannot be connected.

Figure 8.6(d) is particularly interesting because it demonstrates our use of $\mathscr{C}_R^{free}$ connectivity. As opposed to the local planner approach employed in PLR [22], MANIP-SEARCH does not directly attempt to connect two neighboring points in $\mathscr{C}_R$. 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 8.6(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 8.7 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 the couch, however, MANIP-SEARCH finds that it cannot be moved. The planner backtracks, calling RCH again and selects an alternative route.

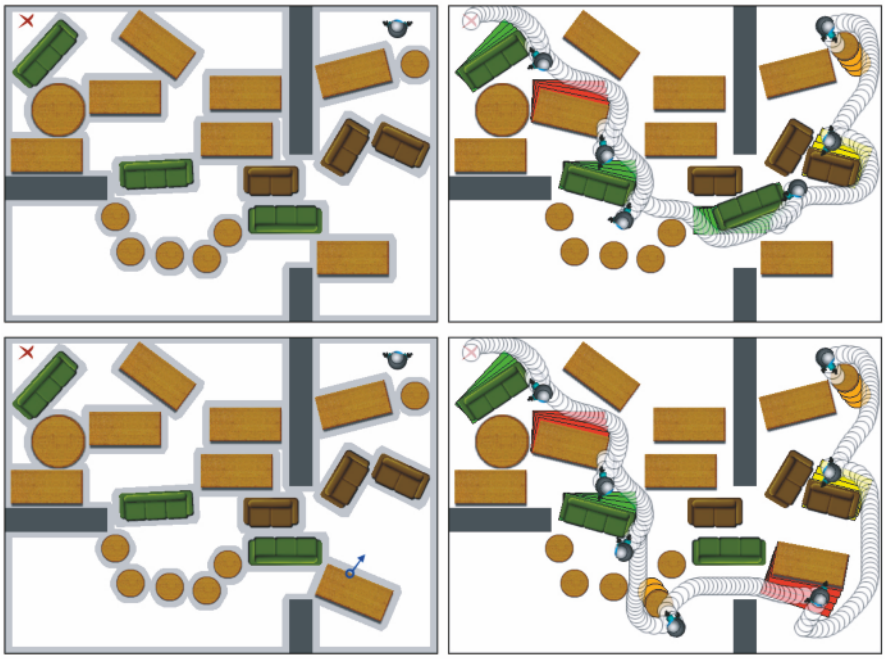


Figure 8.7 The generated plan output by our dynamic simulation NAMO planner is illustrated by the time-lapse sequences on the right

Figures 8.7 and 8.8 show the scalability of our algorithm to problems with more movable objects. While computation time for Figure 8.6(a) is < 1s, the solutions for Figures 8.7 and 8.8 were found in 6.5 and 9s, respectively (on a Pentium 4 3 GHz). 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 with 20 in Figure 8.7, 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 8.2. The solution to (a) is found instantly by the first heuristic search 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.

8.2.6.4 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

Figure 8.8 A larger scale example consisting of 90 movable obstacles. Two separate plans are computed and demonstrated in our dynamic simulation.

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 8.5. *Any solution found by* $\mathrm{SC}(W^t, PrevList, r^f)$ *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 $\mathscr{C}_R^{acc}(W^t)$ (8.13).

Assume that the call to SELECTCONNECT$(W^{t+2}, PrevList, r^f)$ on line 11 returns a valid *FuturePlan*. $\mathrm{SC}(W^t, \ldots)$ 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 (8.2.5.2) and τ_N is valid since $r^{t+1} = \tau_M[0] \in \mathscr{C}_R^{AC}(W^t)$ by construction of MANIP-SEARCH and Definition 8.16. Hence, $\mathrm{SC}(W^t, PrevList, r^f)$ also returns a valid plan. By induction every plan returned by SELECTCONNECT is valid.

Lemma 8.6. *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 8.1 ensures that there exist $r_F \in C_F$ and $\tau_1(r^t, r_F)$ such that for all s, $\tau_1[s] \in \bigcap_k \overline{\mathscr{X}_R(F_k)} \bigcap_{j \neq F} \overline{\mathscr{X}_R^{O_j}(W^T)}$. Let τ_2 be any path in $\mathscr{C}_R$ 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 $\mathscr{C}_R^{free}$ or exclusively in O_F, satisfying the conditions of Definition 8.4. Hence, τ_F is $Valid(O_F, C_F)$. Since a $Valid(O_F, C_F)$ path exists, RCH will find a path (Lemma 8.3).

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 8.1. 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, $x^f \in \mathscr{C}_R^{acc}(W^t)$ 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 Definition 8.3 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 (Definition 8.3). From Lemma 8.6 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 8.4). 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 and the algorithm will terminate.

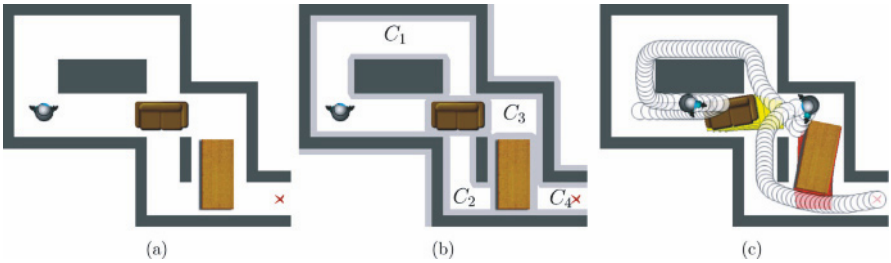


Figure 8.9 SELECTCONNECT solves the open problem considered difficult by Chen

8.2.7 Summary

This section describes initial progress towards planning for NAMO. First, we gave a configuration space representation for NAMO problems. Our analysis of the relationship between action spaces for *Navigate* and *Manipulate* operators gave us tools for constructing a conceptual planner and a practical solution for problems in the intuitive L_1 problem class. Search complexity was reduced from the number of objects in the scene to the difficulty of the *Navigation* task. The planner solved problems with nearly 100 movable obstacles in seconds.

In addition to high-dimensional problems, SELECTCONNECT is also effective in domains with complex geometry. Previously, Chen [22] presented one difficult puzzle problem shown in Figure 8.9. The PLR planner pushes the love seat into C_3 and cannot recover. Using $\mathscr{C}_R^{free}$ 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. Further work on this topic is presented in [23]. Sections 8.3 and 8.4 will show how decisions to move objects are applied in autonomous execution.

8.3 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 section 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 55 kg objects, equal to the robot's mass, can be moved along specified trajectories.

8.3.1 Background

Early results in humanoid manipulation considered balance due to robot dynamics. Inoue [24] changed posture and stance for increased manipulability and Kuffner [25] found collision free motions that also satisfied balance constraints. These methods did not take into account object dynamics. In contrast, Harada [26] extended the ZMP balance criterion for pushing on an object with known dynamics. Harada [27] 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 [28], Takubo [29] applied this method to adapting step positioning while pushing on an object. Nishiwaki [30, 31] proposed that the external forces from pushing could be handled by rapid trajectory regeneration. Yoshida [32, 33] 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 [34] and Fitzpatrick [35] studied impulsive manipulation to detect the affordances of various objects through different sensing modalities. Stoychev [36] considered learning to use objects for specific behaviors and Christiansen [37] 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 [38] summarizes approaches to learning or adapting parameters to achieve precise trajectory following.

Friction modeling has been studied extensively as summarized by Canudas [39] and Olsson [40]. More sophisticated methods for learning the dynamics of tasks in high dimensional spaces are studied by Atkeson, Moore and Schaal [41, 42].

8.3.2 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 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 [43, 44]. 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, it 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 0.15s intervals.

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

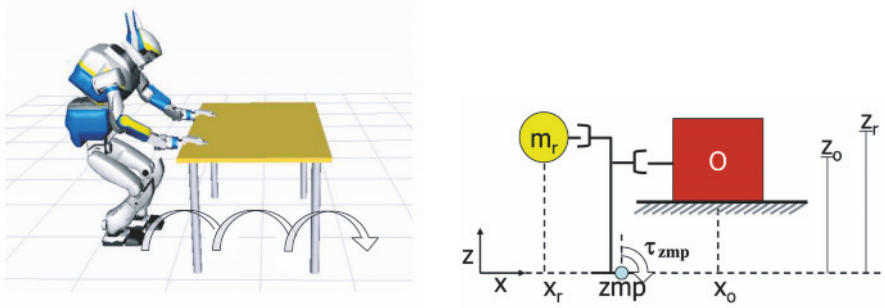
- decoupling the object and robot trajectories;
- trajectory generation satisfying ZMP and object motion;
- online feedback for balance and compliance.

Instantiating these three components lifts control from the 30D robot joint space to a higher level abstract system that realizes a single object trajectory.

8.3.2.1 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 $k = 900$ ms. Section 8.3.2.2 computes independent workspace trajectories for x_r and 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, p_{lh} and p_{lr} remain at their initial displacements from x_o. For simpler analysis, the stance foot p_{st} is fixed relative to x_o at each impact. The robot swing foot, p_{sw} follows a cubic spline connecting its prior and future stance positions. To achieve a fixed displacement



(a) (b)

Figure 8.10 Model of the robot and object used in our work: (a) Geometric model; (b) Computational model

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, p_{torso} relative to x_r. Although the center of mass position, x_r, is a function of all the robot links we assume that x_r remains fixed to p_{torso} after grasp. This assumption is relaxed in Section 8.3.2.2 through iterative controller optimization. Notice that although many of the link positions are highly coupled, the two positions of interest x_r and x_o are not.

Suppose we have workspace trajectories for both x_o and x_r. The former specifies trajectories for hands and feet and the latter defines x_{torso}. Joint values that position the four ungrounded links are found with resolved rate control [45].

$p_{st} \rightarrow x_r$ 6 Stance leg | $x_r \rightarrow p_{lh}$ 7 L arm
$x_r \rightarrow p_{sw}$ 6 Swing leg | $x_r \rightarrow p_{rh}$ 7 R arm

These solutions complete the mapping from any valid workspace placement of x_r and 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 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.

8.3.2.2 Trajectory Generation

Section 8.3.2.1 gave a mapping from commanded workspace positions of x_r and x_o to joint positions. We now focus on workspace control. Given a commanded trajectory for x_o we compute a trajectory for x_r that satisfies balance constraints.

We relate ZMPt 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. Equation 8.20 introduces zmp as the ground point around which the torques due to gravity acting on x_r, reflected

force from accelerating x_r and from the object sum to zero.

$$\tau_{zmp} = m_r g(x_r - zmp) - m_r \ddot{x}_r z_r - z_o f = 0. \tag{8.20}$$

Solving for zmp yields:

$$zmp = x_r - \ddot{x}_r \frac{z_r}{g} - \frac{z_o f}{m_r g}. \tag{8.21}$$

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 x_o (subsection 8.3.2.1)

Let $J_0 = \sum_t (zmp^t - zmp_d^t)^2$ be the performance index for balance. Equation 8.22 further defines β and β_d as functions of zmp_d and x_r respectively.

$$\beta_d = zmp_d + \frac{z_o f}{m_r g} \quad \beta = x_r - \ddot{x}_r \frac{z_r}{g}. \tag{8.22}$$

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.

$$J = \sum_{t=1}^{\infty} Q_e(\beta^t - \beta_d^t)^2 + R(\dddot{x}^t - \dddot{x}^{t-1})^2. \tag{8.23}$$

We can now determine the optimal $\dddot{x}_r$ with preview control [28]. At any time t we know the error $e(t) = \beta^t - \beta_d^t$, state $x(t) = [\, x_r^t \; \dot{x}_r^t \; \ddot{x}_r^t \,]^T$ and N future β_d^i. Stilman [23] gives the procedure for pre-computing the gains G_1, G_2 and G_3 such that the incremental control in Equation 8.24 minimizes J:

$$\Delta \dddot{x}_r^t = -G_1 e(t) - G_2 \Delta x_r^t - \sum_{i=1}^{N} G_3^i (\beta_d^{t+i} - \beta_d^{t+i-1}). \tag{8.24}$$

The control $\Delta \dddot{x}_r$ is discretely integrated to generate the trajectory $\{\ddot{x}_r, \dot{x}_r \text{ and } x_r\}$ for 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 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.

8.3.2.3 Online Feedback

Section 8.3.2.2 described the generation of a balanced trajectory for x_r given x_o. To handle online errors we modify these trajectories online prior to realization with robot joints. Online feedback operates at a 1 ms cycle rate.

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

The trajectory for x_o, or the robot hands, is modified by impedance. We use a discrete implementation of the virtual dynamic system in Equation 8.25 to compute the offset for x_o that results from integrating the measured force error F:

$$F = m_i\ddot{x}_o + d_i\dot{x}_o + k_i(x_o - x_o^d). \tag{8.25}$$

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 10 cm for a 100 N steady state force. This allows for precise trajectory following and soft termination when the trajectory offset exceeds force limits.

8.3.3 Modeling Object Dynamics

Section 8.3.2 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 0.15 s 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.

8.3.3.1 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 500 ms 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. [28] demonstrates that accurate positioning of ZMP requires up to 2s of future information about its placement. Since external forces at the hands create torques that affect the ZMP, they should be taken into account 2s 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.

8.3.3.2 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 Equation 8.26. 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 Equation 8.27.

$$f^t = m_o \ddot{x}_o^t + c\,\dot{x}_o^t. \tag{8.26}$$

$$f^t = c\,\dot{x}_o^t. \tag{8.27}$$

To find 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-2's hands, f^t, at 1 ms intervals. The collected data is represented in Equation 8.28. The term b was used to remove bias, which appeared as a constant force offset allowed by impedance control after grasp.

$$\begin{bmatrix} \dot{x}^1 & \dot{x}^2 & \cdots & \dot{x}^n \\ 1 & 1 & \cdots & 1 \end{bmatrix}^T \begin{bmatrix} c \\ b \end{bmatrix} = \begin{bmatrix} f^1 & f^2 & \cdots & f^n \end{bmatrix}^T. \tag{8.28}$$

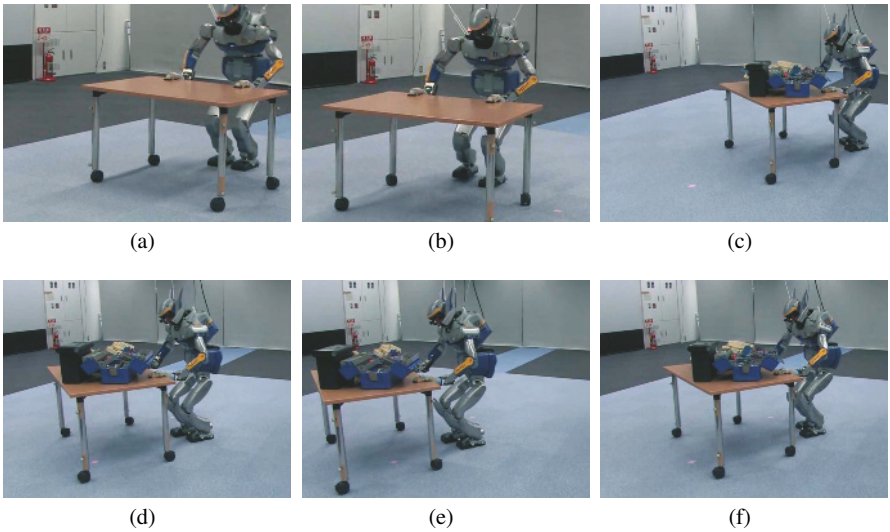


Figure 8.11 HRP-2 pushes then pulls 30 and 55kg loaded tables: (a) t=0s; (b) t=20s; (c) t=0s; (d) t=20s; (e) t=28s; (f) t=38s

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 0.15 s trajectory cycle. This approach was sufficiently stable for brief interactions.

8.3.4 Experiments and Results

We conducted experiments on model-based whole body manipulation using a loaded table on casters, as shown in Figure 8.11. The robot grasped the table and followed a smooth trajectory for x_o as generated from a joystick input. Our results were compared with a rigid grasp implementation of a reactive approach to handling external forces presented in [31], which assumed that sensed forces would remain constant. Both methods recomputed the trajectory for x_r every 0.15 s, 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 8.3.3.2. Brief experiments of less than 10 s 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 30 kg to 55 kg.

8.3.4.1 Prediction Accuracy

First, we look at how well our model predicts force. The comparisons in this section use data from experiments that are not used to build the model. The comparison in Table 8.3.4.1 shows that the mean squared error between modeled and measured force is 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 150 ms 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 8.13 and Table 8.1, also favors the model-based method. x_{err}^{PC} decreases if we assume a faster control cycle. However, even for a 20 ms cycle, we found that error decreases proportionally for both controllers and the ratio of their MSE remains in favor of modeling.

Table 8.1 Prediction Accuracy

	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 8.2 System Stability

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

8.3.4.2 System Stability

The accuracy of prediction has significant effect on the overall stability of the controlled system. Incorrect predictions affect the trajectories for x_r and 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 8.14 shows the increased noise in the control signal for ZMP when using the reactive control. Table 8.2 summarizes this effect.

An even clearer distinction between the two methods is directly reflected in the noise of the perceived force data. Table 8.2 also shows the variance in the noise given the offline filtered signal. The difference in noise is clearly seen in Figure 8.12.

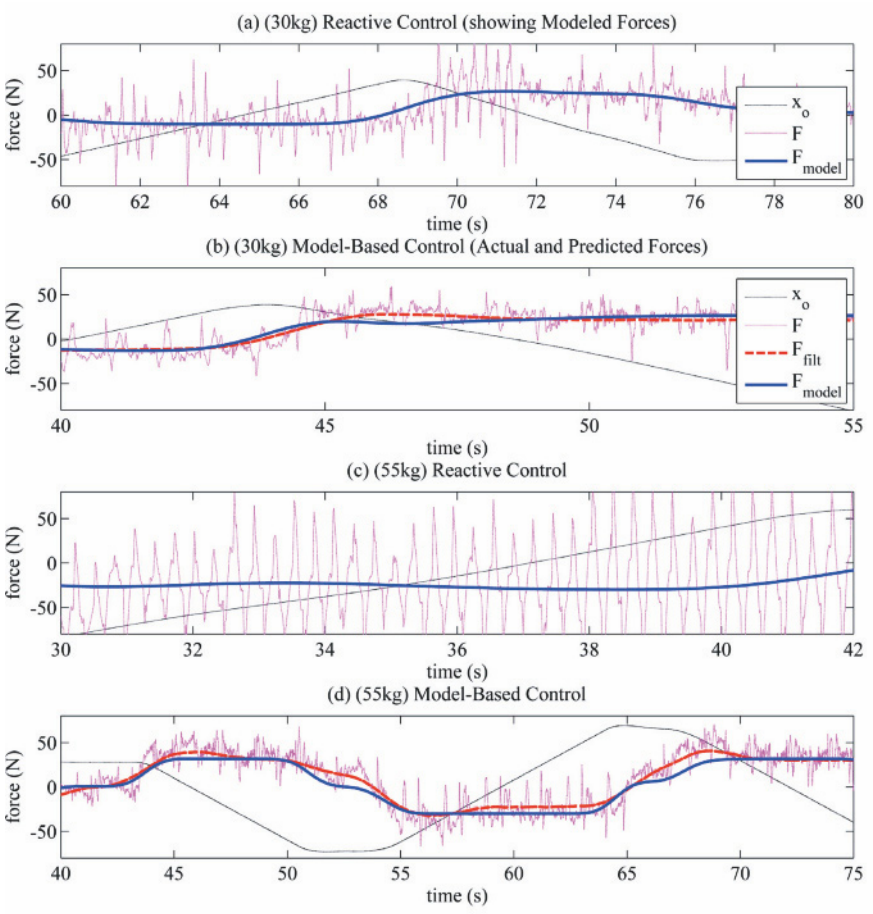


Figure 8.12 Comparison of forces experienced with reactive and model-based control.

8.3.5 Summary

We have shown that it is possible to reliably manipulate unknown, large, heavy objects, such as tables, along specified trajectories with existing humanoid robots. Our experiments demonstrate a significant improvement both in prediction accuracy and system stability when using the learned object model for control. 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. One of the most convincing results is the accurate force prediction for a 55 kg object in Figure 8.12(d). Additionally, notice the low noise variance in sensed forces when using the model based controller.

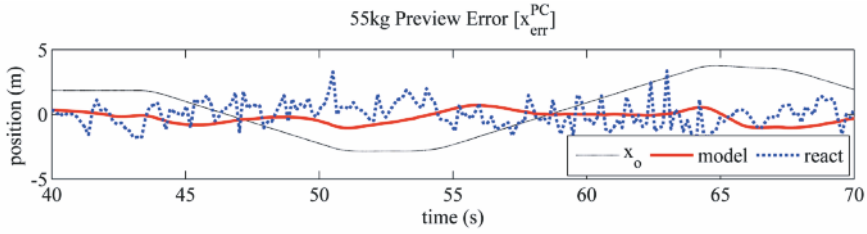


Figure 8.13 Trajectory error introduced by preview control with erroneous prediction

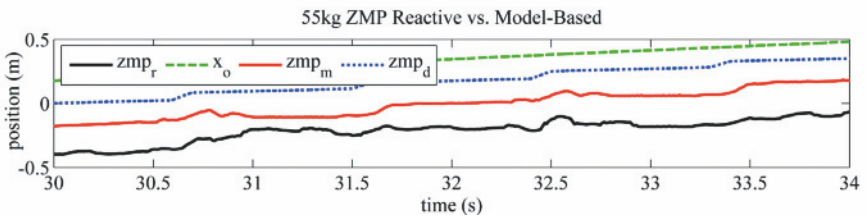


Figure 8.14 Realized ZMP for identical reference trajectories

Future work should consider merging the feed-forward model with feedback 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. The robot can statically estimate the former object mass by measuring the load after lifting. The latter cannot be lifted and would 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. Section 8.4 will detail our complete architecture for NAMO execution.

8.4 System Integration

8.4.1 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 NAMO. This section introduces the architecture for our implementation of NAMO using the humanoid robot HRP-2 shown in Figure 8.15.

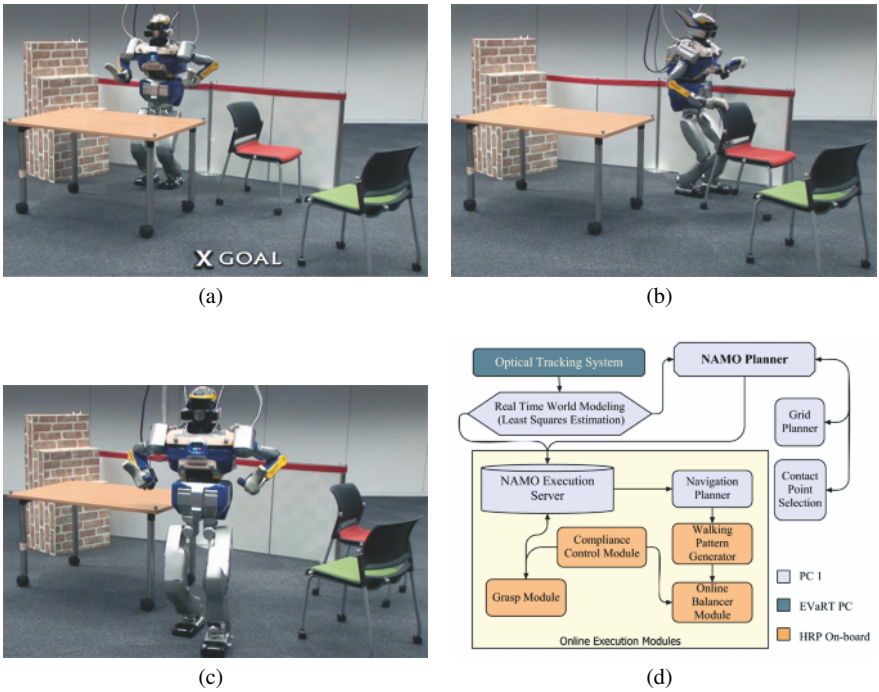


Figure 8.15 Autonomous execution of NAMO with architecture diagram: (a) t=0s; (b) t=45s; (c) t=180s; (d) NAMO Architecture

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 section is derived from Section 8.3. The details of control for robot walking and whole body manipulation were addressed in Section 8.4.

The architecture in this section 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 [46, 47, 48]. Kuffner, Harada, Takubo, Yoshida and Nishiwaki generate dynamically stable trajectories for manipulation given an environment model [26, 49, 31, 29]. Brooks recognizes doors and defines an opening behavior [50]. 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 25 m^2 office setting consisting of tables and 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.

8.4.2 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 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 Section 8.6 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 [51] are summarized by Haralick and Forsyth [52, 53]. 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 [54]. In particular, he advises the use of retro-reflective markers. Some systems use LED markers [55], while others combine vision-based approaches with magnetic trackers [56]. Given our focus on planning and control, we chose an accurate method of perception by combining offline geometric modeling with online localization.

8.4.2.1 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 8.16(b) were edited for holes and processed to minimize the overall polygon count while ensuring that at least one vertex exists in every 0.125 m^3 voxel of the mesh. This simplified object detection in any given region of 3D space to vertex inclusion.

8.4.2.2 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 rigid body assumptions, any 6D 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.

(a) (b) (c)

Figure 8.16 Off-line modeling and online localization of a chair: (a) Chair and markers; (b) Mesh model; (c) Real-time overlay

---

Given positions for unoccluded template, $\{a_1, ..., a_n\}$, and localized markers $\{b_1, ..., b_n\}$:

1. Find the centroids (c_a and c_b) of the template markers points and the observed marker locations. Estimate the translational offset $\hat{t} = c_b - c_a$. Removing this offset, $b'_i = b_i - \hat{t}$ places the markers at a common origin.
2. Next we define a linear system for the orientation of the object,

$$\begin{bmatrix} a_1^T & & 0 \\ 0 & a_1^T & a_1^T \\ a_n^T & \cdots & 0 \\ 0 & a_n^T & a_n^T \end{bmatrix} \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \hat{r}_3 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{bmatrix} \qquad \text{such that} \qquad b_i = \begin{bmatrix} - & \hat{r}_1^T & - \\ - & \hat{r}_2^T & - \; \hat{t} \\ - & \hat{r}_3^T & - \\ 0 & 0 & 0 \; 1 \end{bmatrix} a_i$$

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

Figure 8.17 Object localization procedure.

EVaRT continuously tracks the locations of all the markers to a height of 2 m. Distances between markers are calculated to 0.3 tracking approximately 60 markers the acquisition rate is 60 Hz. 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 8.17 At this time, we do not enforce rigidity constraints. Even for a system with only four markers, the accuracy of marker tracking yields negligible 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 30 hz. Further details and applications of our approach to mixed reality experimentation can be found in [57].

8.4.3 Planning

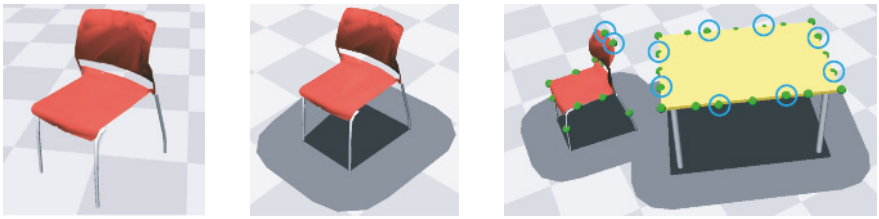
The world model, constructed in Section 8.4.2.2, combined with kinematic and geometric models of the robot is sufficient to implement NAMO planning. However, the search space for such a planner would have 38 degrees of freedom for robot motion alone. The size of this space requires additional considerations which are taken into account in Section 8. Presently, we observe that for many navigation tasks the 2D 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 [26, 49]. 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 Sections 8.3 and 8.4 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.

8.4.3.1 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 8.18 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 0.3 m radius safely encloses torso motion. The cylinder projects to a circle on the ground plane. We pre-compute the navigational configuration space, $\mathscr{C}_R$ of the robot. Each $\mathscr{C}_R$ 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 $\mathscr{C}$-space. Lighter shaded ground regions around projected obstacles in Figure 8.18 are $\mathscr{C}_R$ 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 $\mathscr{C}_R$ obstacles [58]. The resulting obstacles have significantly fewer vertices and edges while subject to only



(a) (b) (c)

Figure 8.18 Mapping objects into the planning domain: (a) Mesh model; (b) Projection; (c) Contact selection

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.

8.4.3.2 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 by interpreting the convex hull from the previous section as a representation of the object perimeter. Starting at an arbitrary vertex of this hull, our algorithm places reference points at equidistant intervals along the edges. The interval is a parameter that we set to 0.2 m.

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 [0.5 m, 1.0 m]. 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 8.18(c). The robot successfully performed power grasps of our objects at the computed locations.

More complex objects may be of width 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. [5] The proposed criteria can still be optimized by using proximity to the contour points as a heuristic for selecting grasps.

(a) (b)

Figure 8.19 Simulated example shows NAMO plan and traces of execution: (a) Initial state and NAMO plan; (b) Traces of execution

8.4.3.3 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 [59, 47]. 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 9 cm 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 $\mathscr{C}$-space obstacles:

1. One 0.1 m translation backward.
2. Twenty rotations in place in the range of 30°.
3. Twenty 0.2 m translations forward with a rotation in the range of 30°.

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 8.19. 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 8.19(b) interpret planned actions as described in Section 8.3.

8.4.4 Uncertainty

The NAMO planner presented in Section 8.2 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 is a complex problem. Erdmann corners the system into a desired state [60]. Stentz efficiently replans while expanding knowledge of the environment [61]. Kaelbling finds optimally successful plans [62]. 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.

8.4.4.1 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 8.3.2.3. This controller handles small positioning errors that occur due to robot vibration and environment interaction, ensuring that the robot does not damage itself or the obstacle. Impedance limits the forces that the robot can exert in order to achieve the precise positioning demanded by the planner.

8.4.4.2 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.

8.4.4.3 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 [63].

The initial workspace hand position for grasping is obtained by selecting a point $.05m$ above the object and 0.05 m 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 0.05 m gaps. This guarded move is executed using impedance control to prevent hard collisions 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.

Table 8.3 NAMO implementation: run times for planning and execution

	NAMO	Navigation	Execution
Figure 8.15 (real)	0.06 s	0.09 s	63.0 s
Figure 8.19 (simulated)	0.13 s	0.93 s	153.0 s

8.4.5 Results

Our complete NAMO system was successfully applied to a number of simulated examples such as the one presented in Figure 8.19 as well as the real robot control problem in Figure 8.15(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 8.3 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.

References

[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] J.C. Latombe. Robot Motion Planning. Springer, 1991.
[3] R. Alami, J.P. Laumond, and T. Sim'eon. Two manipulation planning algorithms. In: workshop on the algorithmic foundations of robotics, 1994.
[4] M.T. Mason. Mechanics of Robotic Manipulation. MIT Press, 2001.
[5] Andrew T. Miller. GraspIt: A Versatile Simulator for Robotic Grasping. PhD thesis, Dept. of Computer Science, Columbia University, 2001.
[6] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. Int J of Robotics Research, 15(6):533–556, 1996.
[7] Michael Erdmann. An exploration of nonprehensile two-palm manipulation. Int J of Robotics Research, 17(5), 1998.
[8] G. Morris and L. Haynes. Robotic assembly by constraints. In: proceedings of IEEE international conference on robotics and automation., 4, 1987.
[9] JD Morrow and PK Khosla. Manipulation task primitives for composing robot skills. In: proceedings of IEEE international conference on robotics and automation, 4, 1997.
[10] M. Stilman and J.J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In: proceedings of IEEE international conference on humanoid robotics (Humanoids'04) http://www.golems.org/NAMO, 2004.
[11] G. Wilfong. Motion panning in the presence of movable obstacles. In: proceedings of ACM symposium on computational geometry, pp 279–288, 1988.

[12] E. Demaine and et. al. Pushpush and push-1 are np-complete. Technical Report 064, Smith, 1999.
[13] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. Artificial Intelligence, 129(1):219–251, 2001.
[14] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In: AIPS98 workshop on planning as combinatorial search, pp 58–60, 1998.
[15] J.C. Culberson and J. Schaeffer. Searching with pattern databases. Lecture Notes in Computer Science, 981:101–112, 2001.
[16] R.E. Korf. Finding optimal solutions to Rubiks Cube using pattern databases. In: proceedings of the fourteenth national conference on artificial intelligence (AAAI-97), pp 700–705, 1997.
[17] L. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning high-dimensional configuration spaces. IEEE Trans Robotics Automat, 12(4), 1996.
[18] S.M. LaValle and J.J. Kuffner. Rapidly exploring random trees: Progress and prospects. In: workshop on the algorithmic foundations of robotics, 2000.
[19] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In: proceedings of IEEE international conference on robotics and automation, 3, 1997.
[20] T. Lozano-Perez. Spatial planning: a configuration space approach. IEEE Trans Comput, pp 108–120, 1983.
[21] S.M. LaValle. Planning Algorithms. Cambridge University Press, 2006.
[22] P.C.Chen and Y.K.Hwang. Pracitcal path planning among movable obstacles. In: proceedings of IEEE international conference on robotics and automation, pp 444–449, 1991.
[23] M. Stilman. PhD Thesis: Navigation Among Movable Obstacles. Technical report, Technical Report CMU-RI-TR-07-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 2007.
[24] K. Inoue, H. Yoshida, T. Arai, and Y. Mae. Mobile manipulation of humanoids: Real-time control based on manipulability and stabilty. In: proceedings of IEEE international conference robotics and automation (ICRA), pp 2217–2222, 2000.
[25] J. Kuffner. Dynamically-stable motion planning for humanoid robots. Autonomous Robots, 12(1), 2002.
[26] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa. Pushing manipulation by humanoid considering two-kinds of zmps. In: IEEE international conference on robotics and automation, pp 1627–1632, 2003.
[27] 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 international conference on robotics and automation, pp 616–622, 2004.
[28] Shuuji Kajita and et. al. Biped walking pattern generation by using preview control of zero-moment point. In: IEEE international conference on robotics and automation, pp 1620–1626, 2003.
[29] T. Takubo, K. Inoue, and T. Arai. Pushing an object considering the hand reflect forces by humanoid robot in dynamic walking. In: IEEE international conference on robotics and automation, pp 1718–1723, 2005.
[30] 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 international conference on Humanoid Robotics, 2006.
[31] K. Nishiwaki and S. Kagami. High frequency walking pattern generation based on preview control of zmp. In: IEEE international conference on robotics and automation (ICRA'06), 2006.
[32] E. Yoshida, I. Belousov, Claudia Esteves, and J-P. Laumond. Humanoid motion planning for dynamic tasks. In: IEEE international conference on Humanoid Robotics (Humanoids'05), 2005.

[33] E. Yoshida, C. Esteves, T. Sakaguchi, J-P. Laumond, and K. Yokoi. Smooth collision avoidance: Practical issues in dynamic humanoid motion. In: proceedings of IEEE/RSJ international conference on intelligent robots and systems, 2006.
[34] E. Krotkov. Robotic perception of material. IJCAI, pp 88–95, 1995.
[35] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through interaction - initial steps towards artificial cognition. In: proceedings of IEEE international conference on robotics and automation, pp 3140–3145, 2005.
[36] A. Stoytchev. Behavior-grounded representation of tool affordances. In: proceedings of IEEE international conference on robotics and automation, pp 3060–3065, 2005.
[37] A. Christiansen, T. M. Mitchell, and M. T. Mason. Learning reliable manipulation strategies without initial physical models. In: proceedings of IEEE international conference on robotics and automation, 1990.
[38] C.H. An, C.G. Atkeson, and J.M. Hollerbach. Model-Based Control of a Robot Manipulator. MIT Press, 1988.
[39] C. Canudas de Wit, P. No, A. Aubin, and B. Brogliato. Adaptive friction compensation in robot manipulators: low velocities.
[40] H. Olsson, KJ Astrom, CC. de Wit, M. Gafvert, and P. Lischinsky. Friction models and friction compensation.
[41] Stefan Schaal Chris Atkeson, Andrew Moore. Locally weighted learning. AI Review, 11:11–73, April 1997.
[42] Andrew Moore, C. G. Atkeson, and S. A. Schaal. Locally weighted learning for control. AI Review, 11:75–113, 1997.
[43] M. Vukobratovic, B. Borovac, D. Surla, and D. Stokic. Biped Locomotion: Dynamics, Stability, Control and Application. Springer, 1990.
[44] M. Vukobratovic and B. Borovac. Zero-moment point-thirty five years of its life. Int J of Humanoid Robotics, 1(1):157–173, 2004.
[45] D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. IEEE Trans on Man Machine Systems, 10:47–53, 1969.
[46] 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. In: proceedings of IEEE/RSJ international conference on intelligent robots and system, 3, 2002.
[47] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami. Planning biped navigation strategies in complex environments. In: 2003 international conference on humanoid robots, 2003.
[48] J. Gutmann, M. Fukuchi, and M. Fujita. Real-time path planning for humanoid robot navigation. In: international joint conference on artificial intelligence, 2005.
[49] E. Yoshida, P. Blazevic, and V. Hugel. Pivoting manipulation of a large object. In: IEEE international conference on robotics and automation, pp 1052–1057, 2005.
[50] 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. Int J Humanoid Robotics, 1(1):1–28, 2004.
[51] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In: proceedings of DARPA Image Understanding workshop, 121:130, 1981.
[52] R.M. Haralick and L.G. Shapiro. Computer and Robot Vision. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1992.
[53] D.A. Forsyth and J. Ponce. Computer Vision: A Modern Approach. Prentice Hall, 2003.
[54] K. Dorfmuller. Robust tracking for augmented reality using retroreflective markers. Computers and Graphics, 23(6):795–800, 1999.
[55] 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.
[56] 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: proceedings of SIGGRAPH'96, page 429, 1996.

[57] 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.
[58] J. Bentley, G.M. Faust, and F. Preparata. Approximation algorithms for convex hulls. Comm. of the ACM, 25(1):64–68, 1982.
[59] JJ Kuffner Jr, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In: proceedings of international conference on intelligent robots and systems, page 500, 2001.
[60] M.A. Erdmann. On Motion Planning with Uncertainty. 1984.
[61] A. Stentz. Optimal and efficient path planning for partially-knownenvironments. In: proceedings of 1994 IEEE international conference on robotics and automation, pp 3310–3317, 1994.
[62] 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.
[63] M. Jeannerod, M.A. Arbib, G. Rizzolatti, and H. Sakata. Grasping objects: the cortical mechanisms of visuomotor transformation. Trends Neurosci., 18:314–320, 1995.